ProDigio™

WSCScript users manual

for WSC31-M / WSC31-C1 / WSC31-T

First Edition

ProDigio Firmware Version 1.07

Programmable Web Server-Module/Chip set

© Copyright 2006 T&D Corporation.Allright reserved.

Feb,2006 16005034101E

Preface

Notices about this User's Manual

- All rights of this manual belong to T&D Corporation. It is prohibited to use, duplicate and/or arrange a part of or whole of this manual without the permission of T&D Corporation.
- All brand names and product names mentioned herein are trademarks or registered trademarks of their respective companies.
- Specifications, design and other contents are subject to change without notice due to continual improvements.
- T&D Corporation accepts no responsibility for any damages arising out of or in connection with the use of this manual.

Getting Information on New Releases of ProDigio

New Release Information about ProDigio is available on the following ProDigio web sites:

ProDigio Web Site	:	http://www.prodigio.jp/
T&D Corporation Web Site	:	http://www.tandd.co.jp/

About this User's Manual

Please be careful with the following characters as they are very similar:
- the numeral "0" and the alphabetical character "O"
- the numeral "1" (one) and the lower case alphabet "l"
- the upper case alphabet "I")

Documentation Conventions

Italics
Please pay special attention to words or phrases written in italics, as important terms or sentences are written in italics.

[Note]

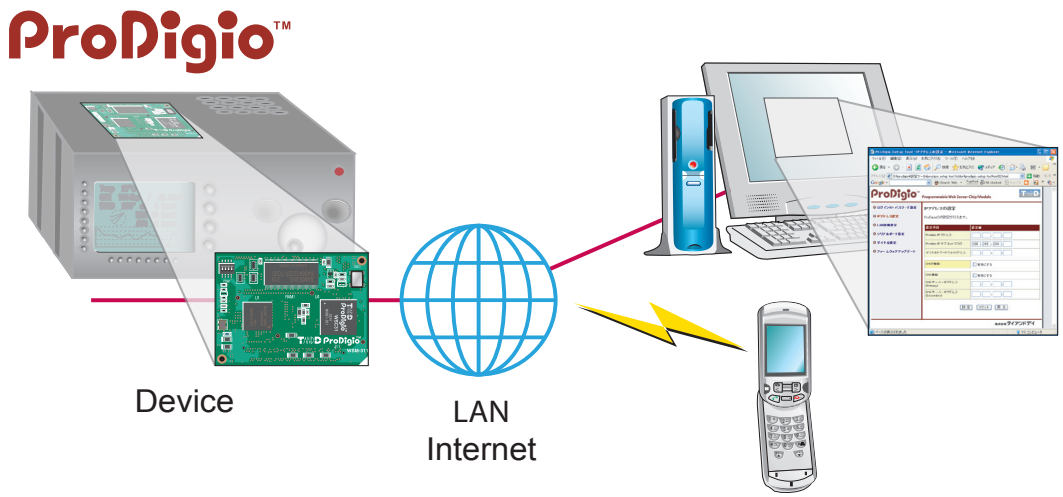
Some explanations and specifications written in this Manual may differ from your actual situation depending on what version of ProDigio Firmware you are using.
This Manual was created for ProDigio Firmware Version 1.07. If you are using a different version from this, some explanations and specifications contained in this Manual may not pertain or may differ from the version you are using.

About ProDigio WSC Script

ProDigio comes with many attractive features that are not supplied by other traditional networking products.

Instant Connection to Network
ProDigio allows you to connect your devices to the network quickly because it provides not only with network interfaces (such as Ethernet card, wireless LAN, data communication card) and peripheral interfaces but also includes all the required drivers. To connect your devices to the network, you need to use ProDigio WSC script and connect your devices with ProDigio

With this script, you can take full advantage of all the features of ProDigio, including data processing, file manipulation, and E-mail delivery. Moreover, the HTTP server feature of ProDigio enables you to significantly extend the network use. For example, you can control your devices or retrieve data from a remote location by using a Web browser and the Internet in conjunction with ProDigio.



Maximize the Functionality of ProDigio Using Script
ProDigio requires some command processing functions to handle requests from the browser. Physically, you can connect your devices to ProDigio through its I/O connectors. However, you need to take some actions for ProDigio to communicate with those devices or control them. To this end, ProDigio provides you with the basic command processing and scripting features and allows you to access peripheral devices via SSI/CGI features.

By programming with WSC script, ProDigio allows you to control your devices and communicate with them at your will in the way that suits connecting devices or meets intended purposes.

This manual provides detailed information on WSC script.

Table of Contents

Preface ---- 1

Getting the Latest Information on New Release of ProDigio ---- 1

About this User's Manual ---- 1

Documentation Conventions ---- 1

About ProDigio WSC Script ---- 2

1. What is a Program? ---- 7

1.1 What is a Program? ---- 7

1.2 Purpose of Programming ---- 7

1.3 What is a Script? ---- 8

1.4 Creating Programs to Instruct ProDigio Work ---- 9

2. WSC Script Standard Specifications 10

2.1 Executing Script ---- 10

2.2 Script Conventions ---- 12

2.2.1 Basic Conventions ---- 12

2.3 Statements and Blocks ---- 14

2.4 Types ---- 15

2.4.1 INTEGER (Numbers) ---- 16

2.4.2 FLOAT (Numbers Containing Decimal Point) ---- 16

2.4.3 STRING (Characters and Strings) ---- 17

2.4.4 Array ---- 18

2.4.5 Index ---- 18

2.4.6 Multidimensional Array ---- 18

2.4.7 Array Operation ---- 19

2.4.8 Array Initialization ---- 19

2.4.9 Defining Array in String ---- 19

2.6 Variables ---- 20

2.7 Understanding Array ---- 22

2.7.1 Variable Declaration ---- 22

2.7.2 Array Declaration ---- 22

2.7.3 Multidimensional Array ---- 23

2.8 Understanding Hash ---- 24

2.9.1 Algebraic Operators ---- 25

2.9.2 Unary Operators ---- 25

2.9.3 Bit Operators ---- 26

2.9.4 Bit OR Operation ---- 26

2.9.5 Shift Operators ---- 27

2.9.6 Comparison Operators ---- 27

2.9.7 Logical Operators ---- 28

2.9.8 Priority of Operators ---- 29

2.10 Expressions ---- 30

2.10.1 Variable Type Conversion ---- 30

3.Control Structure ---- 31

3.1 **if** statement ---- 32

3.2 **else** statement ---- 33

3.3 **while** statement ---- 34

3.4 **do..while** statement ---- 35

3.5 **for** statement ---- 36

3.6 **break** statement ---- 37

3.7 **continue** statement ---- 37

3.8 switch statement ---- 38

3.9 Comments ---- 39

4. Using Escape Characters ---- 40

4.1 \r ---- 40

4.2 \t \b ---- 40

4.3 \\ \" %% ---- 40

4.4 \x ---- 40

4.5 \0 ---- 41

5. Using Functions ---- 42

5.1 Basics of Using Functions ---- 44

5.2 Retrieving Arguments when CGI is in Use - 45

5.3 arg[ ] Array ---- 46

5.4 gethashcount Function ---- 46

6. Working with CGI and SSI ---- 47

6.1. What is CGI? ---- 47

6.2. What is SSI? ---- 48

6.3. Running Script by Form ---- 49

6.4. Running Script by SSI ---- 51

6.4. Running Script by SSI ---- 52

6.5.1. Example of Serial Communication ---- 52

6.5.2. Notes on Occurrence of Other Events ---- 53

6.6. Linking from HTML File to Drive Other Than C Drive ---- 53

7. Time Functions ---- 54

8. File Manipulation ---------- 55
8.1 File Types ---------- 56
8.2 Listing All Files in a Directory ---------- 57
8.2 Listing All Files in a Directory ---------- 58
8.4 Input/Output Binary File ---------- 60

9. Hash Manipulation ---------- 61
9.1 What is a Hash? ---------- 61
9.2 Hash Manipulation Script ---------- 61

10. Using Regular Expressions ---------- 65
10.1 Regular Expression Operators ---------- 65
10.2 Basic Examples ---------- 66
10.3 Advanced Examples ---------- 67

11. Invoking Events ---------- 68
11.1 Registering Events ---------- 68
11.2 Stopping Registered Events ---------- 68
11.3 Autorun Script ---------- 68
11.4 Parameter String (Argument) ---------- 68
11.5 Event Types ---------- 69
11.5.1 TIMER Event ---------- 69
11.5.2 RTC Event ---------- 69
11.5.3 SIO Event ---------- 69
11.5.4 PIO Event ---------- 69
11.5.5 PULSE Event ---------- 70
11.5.6 USER Event ---------- 70
11.6 What is an Event Queue? ---------- 70
11.7 Priority of Events ---------- 71
11.8 Notes ---------- 71

12. Sending E-mails ---------- 72
12.1 Sending E-mails with sendmail Function From Browser ---------- 73

13. About Functions ---------- 75
Layout of Function Specifications ------- 78
13.1 Common Functions ---------- 79
13.1.1 errorR() ---------- 79
13.1.2 istype() ---------- 80
13.1.3 int() ---------- 81
13.1.4 uintstr() ---------- 82
13.1.5 sleep() ---------- 83
13.1.6 hexstr() ---------- 84
13.2 Time Functions ---------- 85
13.2.1 getrtctime() ---------- 85
13.2.2 setrtctime() ---------- 86
13.2.3 getdatestring() ---------- 87
13.2.4 gettimestring() ---------- 88
13.2.5 difftime() ---------- 89
13.2.6 getsntptime() ---------- 90
13.2.7 statsntp () ---------- 91
13.3 String Manipulation Functions 92
13.3.1 chr() ---------- 92
13.3.2 asc() ---------- 93
13.3.3 string() ---------- 94
13.3.4 trim() ---------- 95
13.3.5 len() ---------- 96
13.3.6 strcmp() ---------- 97
13.3.7 strchr() ---------- 98
13.3.8 strrchr() ---------- 99
13.3.9 strstr() ---------- 100
13.3.10 strmatch() ---------- 101
13.3.11 left() ---------- 102
13.3.12 right() ---------- 103
13.3.13 mid() ---------- 104
13.3.14 instr() ---------- 105
13.3.15 tolower() ---------- 106
13.3.16 toupper() ---------- 107
13.4 Hash Functions ---------- 108
13.4.1 gethashcount() ---------- 108
13.4.2 sethashcount() ---------- 109
13.4.3 gethashkey() ---------- 110
13.4.4 sethashkey() ---------- 111

13.4.5 ishashkey() ---------- 112
13.4.6 hashinsert()---------- 113
13.4.7 hashdelete()---------- 114
13.4.8 hashfind() ---------- 115
13.4.9 hashkeysort() ---------- 116
13.4.10 hashsort() ---------- 117
13.4.11 hashcopy() ---------- 118
13.4.12 hashmerge()---------- 119
13.4.13 hashtostring()---------- 120
13.4.14 stringtohash()---------- 121
13.5. Regular Expression Functions 122
13.5.1 regmatch()---------- 122
13.5.2 regmatchall()---------- 123
13.5.3 regreplace() ---------- 124
13.6. File Manipulation Functions- 125
13.6.1 getcurdir()---------- 125
13.6.2 setcurdir()---------- 126
13.6.3 dirmake()---------- 127
13.6.4 dirremove()---------- 128
13.6.5 diropen()---------- 129
13.6.6 dirclose()---------- 130
13.6.7 dirread()---------- 131
13.6.8 filecreat()---------- 132
13.6.9 fileopen()---------- 133
13.6.10 fileclose()---------- 134
13.6.11 fileread()---------- 135
13.6.12 filewrite()---------- 136
13.6.13 filereadhash()---------- 137
13.6.14 filewritehash()---------- 138
13.6.15 getfilesize()---------- 139
13.6.16 filedelete()---------- 140
13.6.17 filerename()---------- 141
13.6.18 filecopy()---------- 142
13.6.19 filetell()---------- 143
13.6.20 filelseek()---------- 144
13.6.21 filechmod()---------- 145
13.6.22 fputs()---------- 146
13.6.23 fgets()---------- 147
13.6.24 fflush()---------- 148
13.7 I/O Functions---------- 149
13.7.1 puts()---------- 149
13.7.2 gets()---------- 150
13.7.3 getchar()---------- 151
13.7.4 beep_on ()---------- 152
13.7.5 beep_off ()---------- 153
13.8. SIO I/O Functions---------- 154
13.8.1 sioputs()---------- 154
13.8.2 siogets()---------- 155
13.8.3 sioputb()---------- 156
13.8.4 siogetb()---------- 157
13.8.5 sioctrl()---------- 158
13.8.6 siorxclr()---------- 159
13.8.7 sioputd()---------- 160
13.8.8 siogetd()---------- 161
13.8.9 siotxclr()---------- 162
13.8.10 siosigctrl()---------- 163
13.8.11 getsiosig()---------- 164
13.8.12 com1_dtr_out()---------- 165
13.8.13 com1_ptn_in()---------- 166
13.8.14 com1_ctrl()---------- 167
13.9 PIO I/O Functions---------- 168
13.9.1 putport()---------- 168
13.9.2 getport()---------- 169
13.9.3 putpio()---------- 170
13.9.4 getpio()---------- 171
13.9.5 setport()---------- 172
13.9.6 setpio()---------- 173
13.9.7 getpiopulse ()---------- 174
13.9.8 clrpiopulse ()---------- 175
13.9.9 statport ()---------- 176
13.9.10 statpio ()---------- 177
13.9.11 putexport()---------- 178
13.9.12 getexport()---------- 179
13.9.13 putexpio()---------- 180
13.9.14 getexpio()---------- 181
13.9.15 setexport()---------- 182
13.9.16 setexpio()----------183

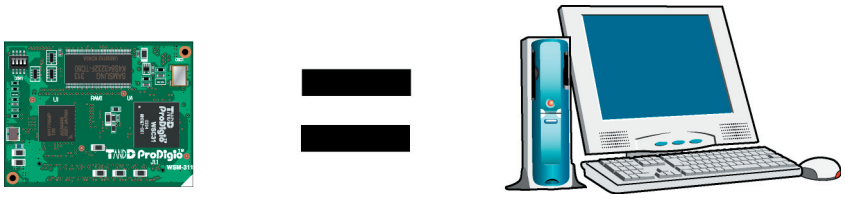
13.9.17 statexport () ---------- 184
13.9.18 statexpio ()---------- 185
13.10 AD Conversion Functions--- 186
13.10.1 getadc()---------- 186
13.11 I2C I/O Functions---------- 187
13.11.1 i2cput() ---------- 187
13.11.2 i2cget() ---------- 188
13.11.3 i2cset() ---------- 189
13.11.4 i2cwrite()----------190
13.11.5 i2cread() ----------191
13.12 FTP Send/Receive Functions192
13.12.1 ftpput() ---------- 192
13.12.2 ftpget() ---------- 193
13.12.3 statftp()---------- 194
13.13. Send E-Mail Functions ------ 195
13.13.1 sendmail()---------- 195
13.13.2 stat_smtp()---------- 196
13.14. Web Output Function -------- 197
13.14.1 webprint()---------- 197
13.15. Event Process Function ---- 198
13.15.1 event()---------- 198
13.16 System Control Functions -- 201
13.16.1 reboot() ---------- 201
13.16.2 getwscparam() ---------- 202
13.16.3 setwscparam() ---------- 203
13.16.4 storewscparam()----------204
13.16.5 crtwscparam()---------- 205
13.16.6 reiniwscparam()---------- 206
13.16.7 getmacaddr() ---------- 207
13.16.8 phpowctrl() ---------- 208
13.16.9 get_module_sw()----------209
13.16.10 outeventlogmsg()----------210
13.16.11 getsysver()----------211
13.17 Dialing Control Functions--- 212
13.17.1 start_atdialup()---------- 212
13.17.2 stop_atdialup() ---------- 214
13.17.3 stat_atconnect() ---------- 215
13.17.4 setini_atcommand()----------216
13.17.5 setdial_atcommand()----------217
13.17.6 get_atispno() ---------- 218
13.17.7 get_atresult() ---------- 219
13.17.8 seterr_atresult ()----------220
14. Appendix ---------- 221
14.1 Notes on the Use of Script ---------- 221
14.2 List of Error Messages ---------- 222
14.3 List of Startup Parameters ---------- 223
14.4 List of Network Parameters ---------- 224
14.5 List of Serial Parameters ---------- 226
14.6 List of Other Device Parameters---------- 228
14.7 Environment Variable Parameters--------- 230
14.8 Dialing Control Parameters ---------- 231
14.9 Available Regular Expressions---------- 232
14.10 Detailed Information about Error Codes ----
---------- 233
14.11 Terminal Software---------- 235
14.11.1 Connecting to Hyper Terminal----------235
14.11.2 Creating New Connection ----------236
14.11.3 Closing Hyper Terminal----------239
14.12 List of Debug Monitoring Commands--- 240
14.13. Debug Monitoring Functions ---------- 241
14.13.1 Standard Output ----------241
14.13.2 Error Messages ----------241
14.13.3 Event Log----------241
14.13.4 Referring to WSC Configuration Parameters ---------
----------242
14.13.5 Connecting to Debug Port via Telnet ----------242
14.13.6 List of Debug Error Messages----------242
Index---------- 244
Support and Customer Service-------- 248
About Us ---------- 248

1. What is a Program?

1.1 What is a Program?

When you are using a computer, you are also using software.
A computer does not work just by configuring hardware; it requires software to process various tasks. Any application software (such as an operating system, word processor, or spreadsheet) and drivers that you use in your routine work consist of programs. When using a word processor application or a spreadsheet application, you are making the computer perform word processing tasks or table calculation tasks. A program is a list of instructions telling the computer how to perform those tasks.

ProDigio, although very small, is with no doubt, it is a type of computer



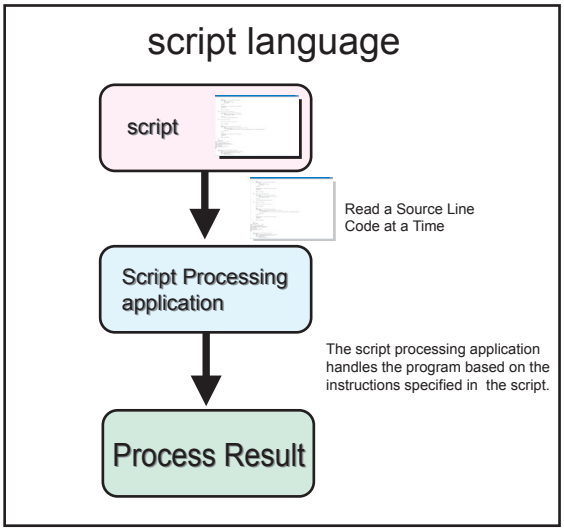
1.2 Purpose of Programming

ProDigio can be used for various purposes including:
- Temperature/ Humidity Control
- Regular remote viewing of home security camera images
- Remote verification of the progress of a manufacturing line

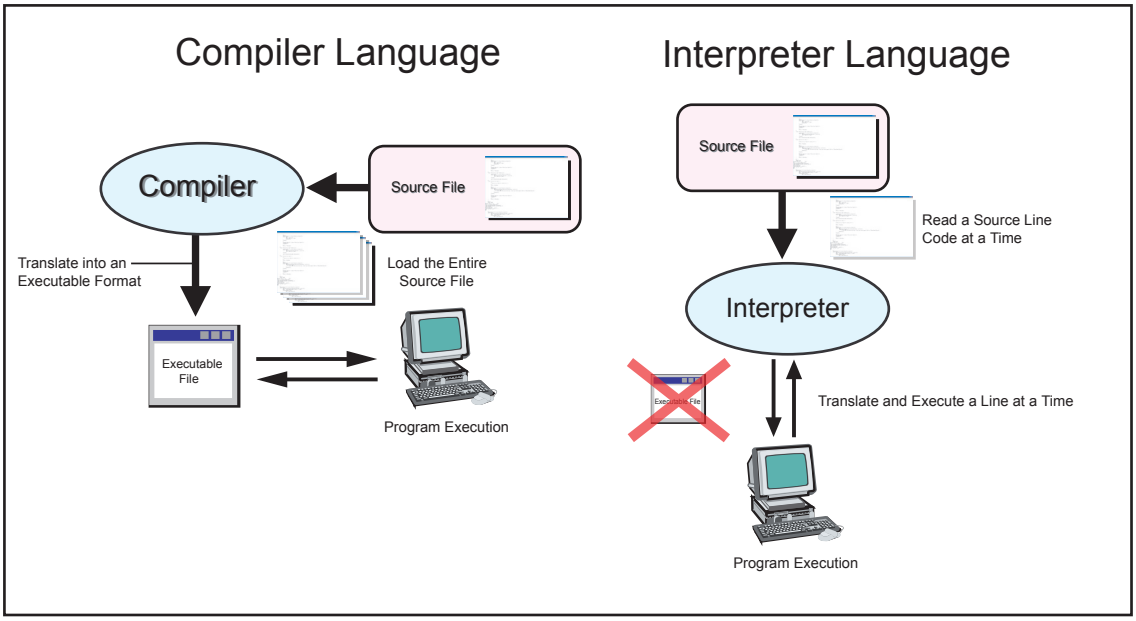
To use ProDigio for any specific purpose, you need to create a program for that specific purpose. It is natural for you to think that programming is not an easy task to accomplish. However, because ProDigio has basic software (OS and various types of drivers) built-in, all you need to do to make the most of ProDigio features is to write a simple program to meet your requirements.

1.3 What is a Script?

The machine language is a sequence of 0's and 1's and it is the only language that computers can understand.
On the contrary, it is a very difficult language for us humans to understand. To help solve this difficulty, we designed a way of writing programs with a programming language that is closer to a human-understandable-language. Then, that can be translated into a machine language that computers can understand.
A script is a programming language that is ready to be translated automatically into machine language.



ProDigio WSC script uses interpreter software that translates script statements line-by-line as the program is running.



1.4 Creating Programs to Instruct ProDigio Work

We can ask other people to do something just by asking them in a vague way such as: “Could you please do that like this?”
However, computers are not like humans. They do not work properly with such unclear instructions and need every single step to be specified.
For example, if you want ProDigio to display a list of directories onto the screen, you need to use the following steps:

Tell it the directory number.
↓
Tell it the file name.
↓
Tell it the name of the directory to open.
↓
Tell it to open the directory

Below are specific steps to tell ProDigio to load the text file in C:¥www¥cgi-bin¥sample.txt:

Tell it that the location and the name of the target file is C:¥www¥cgi-bin¥sample.txt.
↓
Tell it to obtain the size of the file.
↓
Tell it to load the file.

It is necessary to give ProDigio detailed instructions during each step of the process as in the above instructions.
As explained in the previous section, ProDigio operates based on the instructions (orders) written in the script language. The script language is indeed a language. Therefore, it has rules you need to follow in terms of character strings (such as words, numerical value, punctuation, and various symbols), grammar, and syntax.

2. WSC Script Standard Specifications

As described in the previous chapter, ProDigio communicates with various devices and controls them using scripts that you have created. This chapter starts with introducing the basis of WSC script.

2.1 Executing Script

Before going into the details about WSC script, let's first actually execute a script and experience how it works.
It is assumed that you are using the Training Board "WSC31-T

Executing WSC Script in a Module

Creating WSC script for ProDigio involves the task of writing code with a script language.
In this section, you will create a simple program that displays a string by using terminal software. Before you begin, please set up your computer and a module so that you are ready to run script and see the results. For more information about setting your computer and the development module, see the "Training Kit User's Manual."

1. Write a script on your computer.
   Open a text editor. If you are using Windows, you can use Notepad.

2. In the text editor, enter:

```
Function main()
{
        puts("HELLO WSC\n") ;
}
```

3. Save the script file you created as "test.wcr".

4. Transfer the file to a WSC module.
   Use **FTP** or USB to transfer test.wcr over to the WSC module.
   The destination directory is "C:¥www¥."
   For more information on how to transfer files with **FTP** software and connecting with ProDigio via USB, refer to the "Hardware User's Manual."

5. Run the script.
   Start any terminal software such as HyperTerminal on your computer, and establish a serial connection. Refer to the "Hardware User's Manual" for information on establishing a serial connection.

6. Enter the following line of text in the terminal software window.

```
> cd "C:\www"
```

7. Next, run the command.

```
> exec "test.wcr"
```

8. You should see “HELLO WSC” in the terminal software window as follows:

```
> exec "test.wcr"
HELLO WSC
```

This is how to write statements in WSC script that makes ProDigio display character strings in a terminal software window. This is a very simple operation, however, programs provided in this document hereafter are made up of this sort of simple statements.

To put it simply, this series of statements tells ProDigio to output the specified string by using **puts** function and enclosing the string to be displayed (which is “HELLO WSC”) within the subsequent brackets.

All of these statements (functions) are listed in “13. About Functions” of this manual. Please refer to the description of each function to learn about its meaning, role, and usage.

2.2 Script Conventions

In the previous section, as an introduction to WSC script, you ran the script that displayed a character string onto the screen.
To write script for ProDigio to work in a desired way, you cannot just enter statements, but you must comply with various script conventions.
This section provides information on the script conventions (rules.)

2.2.1 Basic Conventions

• Basically, write script in descending order of processes.
• Enclose a group (block) of processes in a pair of curly brackets ({ }.)
• Separate processes by semicolons (;).
• There is no limitation on the use of spaces, line breaks, and tabs between functions (statements) as long as the script does not exceed the maximum size.

Although there are many other conventions to learn, you should start with the basics first and gradually move on to more advanced conventions. Following is an example of a program that has ProDigio perform addition:

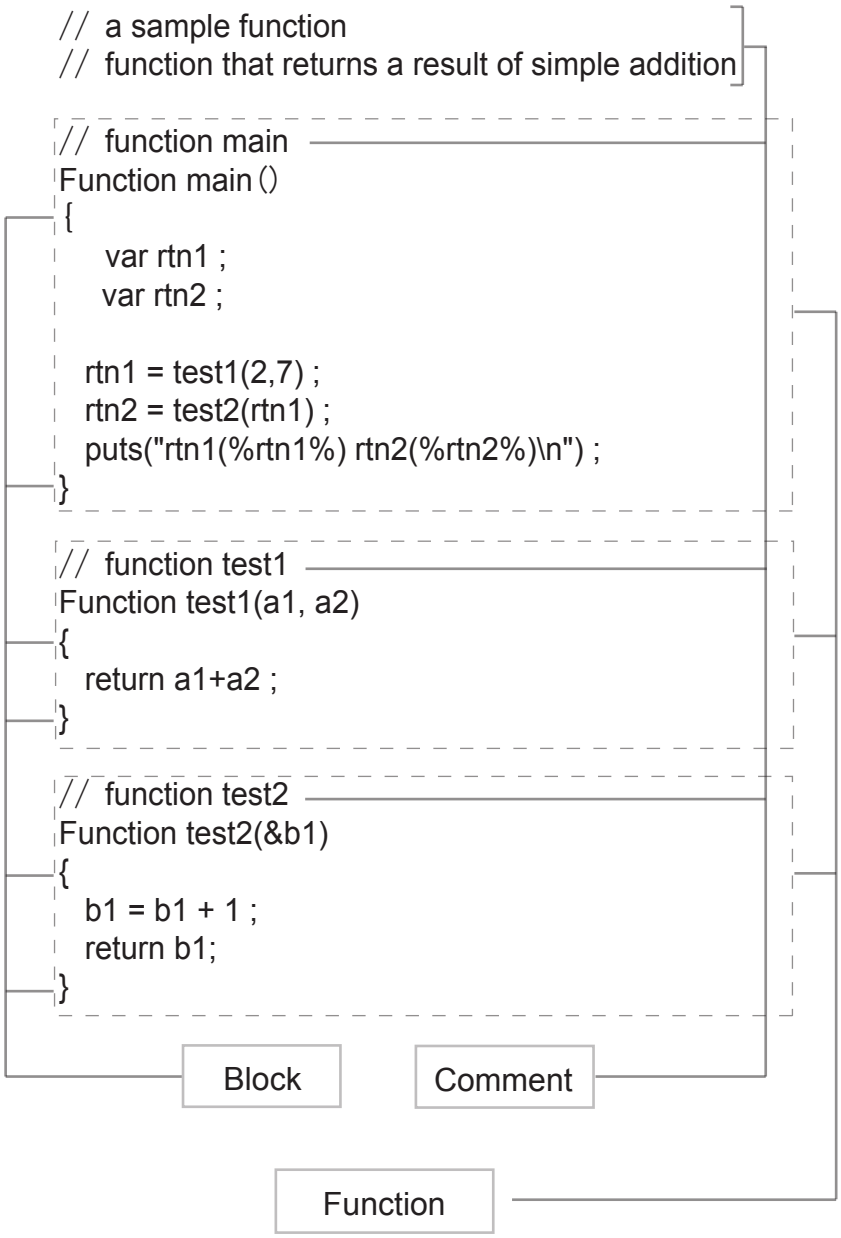
```
// a sample function
// function that returns a result of simple addition
// function main

Function main()
{
var rtn1 ;
var rtn2 ;

rtn1 = test1(2,7) ;
rtn2 = test2(rtn1) ;
puts("rtn1(%rtn1%) rtn2(%rtn2%)\n") ;
}

// function test1
Function test1(a1, a2)
{
return a1+a2 ;
}

// function test2
Function test2(&b1)
{
b1 = b1 + 1 ;
return b1;
}
```

Result of script execution

rtn1 = 10

rtn2 = 10

Result of Script Execution
It may be difficult for you to figure out the roles of each statement just by looking at this sample. The below illustration will help you understand the roles of each statement:



As shown in the above diagram, script can be divided in to three parts.
The first part called “Function” contains various directive text and works as a statement.
The next part called “Code Block” plays the role of bunching multiple statements into a process within a function or across functions.
And, the third part called “Comment” has nothing to do with the behavior of the program and you can write anything to help others understand what is going on in the program. It is recommended that you write comments for later reference so that you will know the purpose of statements.

You are probably aware that this code snippet contains some blanks (spaces) in places. They are SBCS spaces and tabs used for the sake of readability. You can move these SBCS spaces and tabs, as you like for the ease of reading script.

2.3 Statements and Blocks

We use “sentences” and “paragraphs” when writing. Likewise, script has “statements” and “blocks.”
A “statement” used in the script is an instruction to the computer.
When writing statements in WSC script, ProDigio does not identify the end of a statement just by pressing the **Enter** key, you must enter a semicolon (;) at the end of each statement as you would use a period at the end of a sentence.

[Sample Statement]

```
return a1+a2 ; <-- Remember to place a semicolon at the end of a statement.
```

A block is like a paragraph grouping these statements together and segmenting statements within a program.
Please note that a block has to be made up of at least one statement.

Keep in mind that you enclose statements in a pair of curly brackets ({ }) when creating a block.

[Sample Block]

```
Function test1(a1, a2)
{ <-- Beginning of a Block
return a1+a2 ;
} <-- End of a Block
```

You can also include a block inside of another block to enclose multiple instructions in a block.

2.4 Types

A type is the abstract representation of a variable or an object for the computers to understand what it is.
For example, when you want the computer to process numbers, you need to tell it that they are numbers and what kind of numerical format they use.

In most cases, WSC script accepts the following types:

- INTEGER (used for any numerical value that does not contain decimal points, including decimal numbers, octal numbers, and hexadecimal numbers)
- STRING (used for Hiragana, DBCS/SBCS Katakana, Kanji, DBCS/DBCS alphabet)
- FLOAT (used for any numerical values containing decimal points)

Notes:

Binary Number: a number in base two and only has values of 0 and 1. (0.1.10.11.100.101.110.111.1000...)
One hundred as a decimal number is 1100100 as a binary number.

Octal Number: a number in base eight and has values of 0 - 8. (5.6.7.8.10.11...17.18.20...)
One hundred as an octal number is 64 as a decimal number.

Decimal Number: a number you usually use for counting. The number is in base 10 where the second digit increments when the first digit goes a round of zero to nine. (5.6.7.8.9.10.11...)

Hexadecimal Number: a number in base 16 that uses numbers from zero to nine and alphabets from a to f.
(8.9.a.b.c.d.e.f.10.11....)

In this way, variables can be split up into three types.
In regard to types of "*variables*"[*1], you can write a very flexible script.
For instance, assume that there is a number declared as a type of INTEGER. When you instruct the number to be treated as a type of STRING by using a variable, the value will be converted into a type of STRING.
In other words, a "type" is a changeable entity. It does not have to stay as is and can be changed depending on the situation.

[*1] Variables

See "2.6 Variables" for more information.

2.4.1 INTEGER (Numbers)

This type accepts only signed 32-bit numerical values[*2], [*3] but is capable of handling decimal numbers, octal numbers, and hexadecimal numbers.
The first term used in the following statements, “val”, is an indication of the variable (explained later in this chapter.)

• **Decimal Notation**

```
val = 1234;      // decimal number
val = -123;      // negative number
```

• **Octal Notation (with a leading zero)**

```
val = 0123;      // octal number (equivalent to decimal number 83)
```

• **Hexadecimal Notation (with a leading zero)**

```
val = 0x12;      // hexadecimal number (equivalent to decimal number 18)
```

[*2] Range of Available Numbers
Hexadecimal Number: between 0x80000000 and 0xffffffff
Decimal Number: between -2147483648 and 2147483647

[*3] Bit
A bit is the smallest unit of information that a computer can process and it is also the amount of information required to make a choice between the two. A byte is a sequence of 8 bits processed as a single unit of information.

2.4.2 FLOAT (Numbers Containing a Decimal Point)

The type FLOAT only supports single-precision floating-point numbers.
For a number of type FLOAT, **up to the 6th decimal place will** *hold good* but the 7th place will be rounded to the millionth.
An example of rounding is “f=13.0/7=1.857143.”

[Example of a Type FLOAT]

```
fval = 0.00010;
```

2.4.3 STRING (Characters and Strings)

This type of value needs to be enclosed in double quotation marks (“) and this rule applies to any case where afterwards you will treat a value as a string.

• **Joining Strings (+)**

Addition of strings results in the operation of joining strings.

Example: str = “abc” + “def” ; //”abcdef”

The above operation sets the value of string variable **str** to “abcdef.”

• **String Expansion (%Variable%)**

When you specify *%Variable name%* within a string, the value of the variable is expanded (substituted) as a string.

Example: val = 1 + 2 ;

str = “1 + 2 = %val%” ;

The above operation sets the value of the string variable **str** to “1 + 2 = 3.”

• **Escape Characters**

The following table lists the escape characters used in WSC script:
Detailed information on escape characters is provided in “4. Using Escape Characters” of this manual.

Escape Character	Description
\n	Line break
\r	Carriage return
\t	Horizontal tab
\b	Backspace
\\	\
\"	"
%%	%
\ooo	Octal notation
\xhh	Hexadecimal notation

Note: Escape characters are used in place of non-literal characters such as line breaks, spaces, tabs, and so on. For example, type “\n” to insert a line break in-between letters/terms displayed on the screen.

[Example of Using Escape Characters]

```
Function main()
{
   puts("HELLO\n") ;
   puts("This is WSC-31 Web Server Module\n") ;
}
```

Line Breaks

2.4.4 Array

An array is a sequential collection of INTEGER or STRING data types described in the previous section.
Data in an array is called an “element” and these elements are identified by index (suffix.)
An array is one of the basic data format used in the world of programming languages. However, WSC script is different from other typical programming languages in that its array can store not only one data type of variables but also a **mixture of data types**, whereas most programming languages can load into memory only one type of data specified by **index** (suffix).

Another different point is that, in WSC script, **the maximum size of an array changes dynamically, depending on the amount of memory** being used. Therefore, you will not be able to use a very large array.

The general idea of an array is described in “2.7 Understanding Array.”

[Example of Array]

```
a[0] = "abc";
a[1] = "def";
b["foo"] = 13;
```

2.4.5 Index

Accessing the suffix with a numerical value means that you are accessing the array via index. The index starts with zero and the limit on the size of data storage in an array is determined based on the amount of memory installed on the computer.
Accessing the suffix with a string means that you are accessing the array via a key of associative array (hash)[*4] and the size of the array will be automatically enlarged.

[Example of Specifying Index]

```
var  a[100];          // prepare 101 arrays
var  a["100"];        // prepare an array with a key of “100”
```

[*4] Associative Array (Hash)
Details of associative arrays (hash) are described in “2.8. Understanding Hash.”

2.4.6 Multidimensional Array

A **multidimensional array**[*5] is an array of arrays and has another [key] at the end of each dimension.

[Example of Multidimensional Array]

```
var  a[1][0]     = f;     //2two-dimensional array
var  a["foo"][2] = f;     //ordinal index & associative suffix can be used together
```

[*5] Multidimensional Arrays
For more information on multidimensional arrays, see “2.7.3 Multidimensional Array.”

2.4.7 Array Operation

To assign or bind arrays, you need to provide an array without index.

[Example]

```
a[ ] = b[ ];            // copy array b to array a
a[ ] = b[ ] + c[ ];     // bind array b and array c, and copy it to array a
```

2.4.8 Array Initialization

An array is initialized within a declaration statement with a list of elements enclosed in a pair of {}.

[Example]

```
a[ ]={1,2,3};                          // equivalent to a[0] = 1;a[1] = 2;a[2] = 3;
a[ ]={a = 10, b = 20};                 // equivalent to a[“a”] = 10;a[“b”] = 20;
a[ ]={1,2,{10,20,{"a","b","c"}},3};    // initializing a multidimensional array
a[ ]=0;                                // equivalent to a[ ] = 0[ ].
                                       // emptying the contents of an array
b = {"A", "B", "C"}[1];                // assign “B” into b
```

“2.7.3 Multidimensional Array” provides more detailed information on the above example by using a similar script and shows how data is stored in an array.

2.4.9 Defining an Array in a String

To place an associative array inside a string, you need to use [\”] as a double quotation mark instead of using [“].

[Example]

```
str = "%something[\"ab\" + i]%";
```

Note:

Do not use the same name for the Variable and the Array.
It may result in an a variety of trouble.

2.6 Variables

A variable is a place where a value is changeable or a storage location for a value or string. As described in the previous section, computers handle all data in numerical form.
Any computer has memory installed and stores numerical form of data in memory; in other words, any data will be stored in numerical form regardless of its type, STRING or INTEGER.
Computers on their own do not have any knowledge or ability to deal with strings. It is the list of character codes that provides the mechanism for displaying and recording characters that we recognize. This list contains characters used on computers with each character having a unique number assigned.

For example, a string of “Hello” is converted into the following numerical values on computers:

Hello (in ASCII Code Representation)					
	H	e	l	l	o
Hexadecimal Notation	48	65	6c	6c	6f
Decimal Notation	72	101	108	108	111
Octal Notation	110	145	154	154	157
Binary Notation	01001000	011000101	01101100	01101100	01101111

Following is the internal procedure performed to display a character “H” on to the screen: a. the character code of “H” is 0x2433 in hexadecimal notation; b. the computer is instructed to display a character that corresponds to 0x2433 in the list of character code.
In this way, computers handle everything as “numerical data” even though we identify them as “characters.”

A “variable” is allocated in memory and it is like a box stored in a warehouse where you can store strings and numbers.
Memory allocated variables (boxes) can hold almost anything and data is retrieved as needed. In addition, you can “substitute” values to variables with any name of choice like x or y, as with those used in mathematics.

[Example of Using Variables]

```
// function main
Function main()
{
 var rtn1 ;
 var rtn2 ;
 rtn1 = 10 ;
 rtn2 = 100 ;
 puts("rtn1(%rtn1%) rtn2(%rtn2%)\n") ;
}
```

The name of a variable must *be declared with* ***var*** in advance.
Imagine that there is a warehouse called memory with some boxes inside of it; in this situation, declaring variables is like specifying numbers and names to empty boxes. Once a variable is used, its content is stored in memory and initialized with zero.
In the above example, statements starting with **var** tell computers that variables are going to be used in the subsequent code. This kind of statement to let computers know what is going to happen is referred to as a "declaration."
Next come the names of boxes to store data. In this example, two boxes named "rtn1" and "rtn2" are provided in the memory. The subsequent two lines are instructing to substitute 10 and 100 to **rtn1** and **rtn2** respectively.

You cannot expect this rule applies to all cases, but in most cases, it is required to declare variables before executing any statements, that is at the very beginning of the code.
Any variables declared outside of a function is called a global variable and this is a variable that is declared for the entire script; therefore this type of variable can be referenced or updated anywhere in the script.
On the other hand, any variable declared within a function is called a "local variable" and this type of variable stays alive only within the scope of the function where the variable is declared. As global variables are declared when the script is loaded, they stay in memory while the process of the script is running (until the end of script process.) On the contrary, local variables are released (cleared) from memory when processing of the function in which they are declared terminates.

Please keep in mind as a note on variable name conventions that variable names may contain alphanumerics and underscore (_), however, they cannot start with a number and are not case-sensitive.

[Difference between Global Variables and Local Variables in Script]

```
var  G_1;

Function main()
{
        var  M_1;
        var  M_2;
}

Function func()
{
        var  F_1;
        var  F_2;
}
```

- The variable G_1 is a global variable that can be used both in **main()** and **func()** functions.
- The variable M_1 and M_2 are local variables that can be used only within the scope of **main()** function.
- The variable F_1 and F_2 are local variables that can be used only within the scope of **func()** function.

2.7 Understanding Arrays

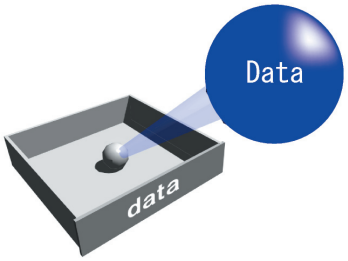
The previous section described how “variables” can be used as a box. This section will provide information on “arrays” where you can store many boxes in a line.

A variable can hold a piece of data.
An array is a form of data where various types of data are stored in sequential order.
Data in an array are called elements and they are identified by a suffix[*1].
An array is one of the basic data formats used in most programming languages.
It stores various types of variables in memory, but only for the number of variables specified by the index (suffix) and reads/writes data from there. It is also possible to perform batch processing by specifying a variable as an index and using a loop[*2].

Some languages are capable of handling “multidimensional arrays” where you can have a child array. ProDigio WSC script is one that can support multidimensional arrays.

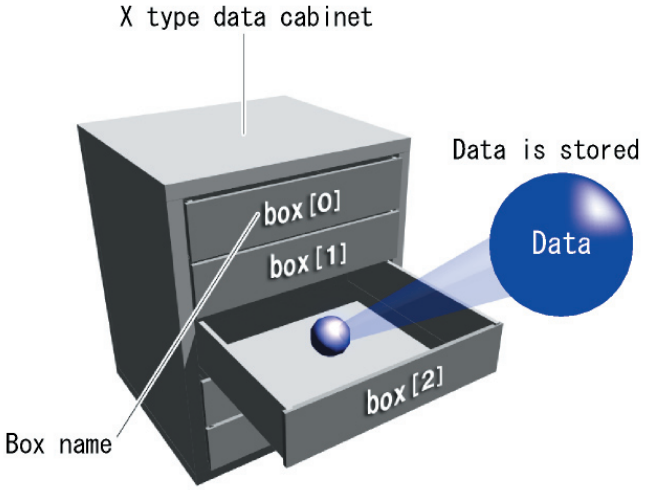
2.7.1 Variable Declaration

The declaration of a variable such as **var data**; (declaring a variable of INTEGER type, named data) creates a box labeled “data” as shown in the following diagram:



2.7.2 Array Declaration

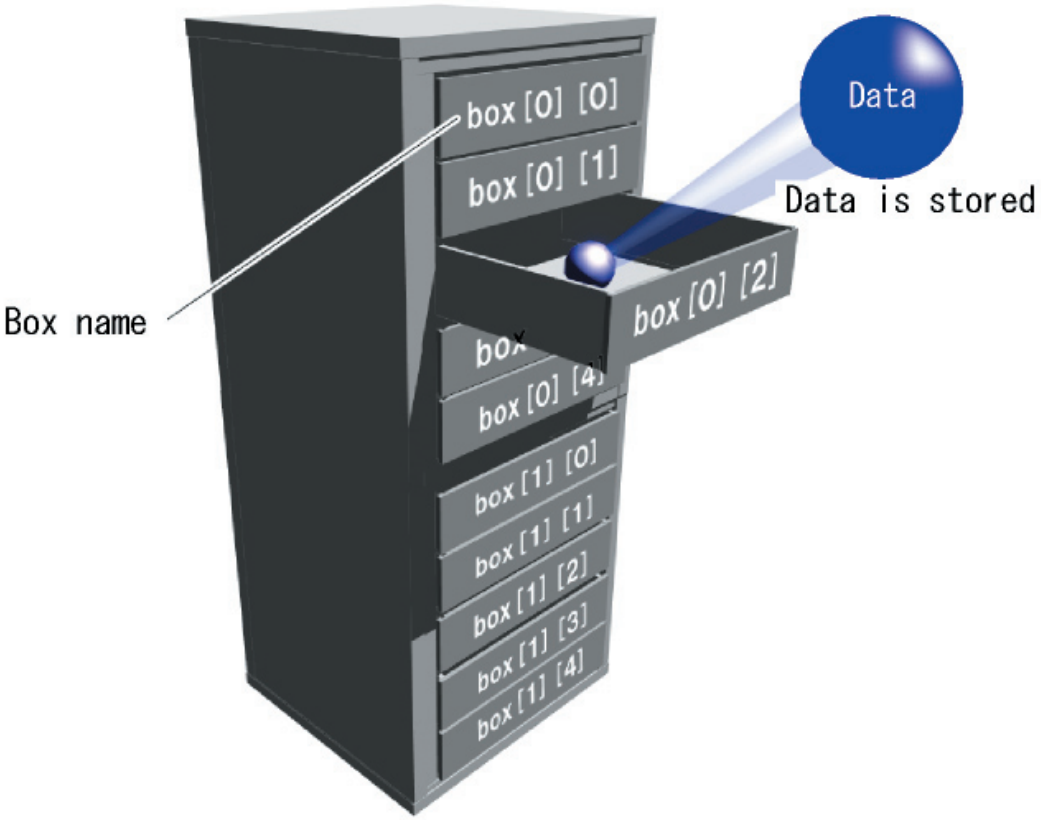
The declaration of an array such as **var box[4]**; (declaring an array of INTEGER type, named box, with 5 elements) creates a box with 5 drawers for storing data of INTEGER type as shown in the following diagram. And each drawer is labeled as box[0] - box[4] respectively.



2.7.3 Multidimensional Array

A multidimensional array is an array of arrays.
The following statement declares an array that has two elements, each having an array of five elements.
It is like a box containing two boxes with 5 drawers.

[Declaration Statement]
box[2][4];



*1 Suffix

This is a number (also known as an index), indicating the position of an element in an array and the number starts with zero. In the example of this section, [0] of box[0] is a suffix.

*2 Loop

A process of repeating the same process.

For instance, you would use a loop when you need to perform an iterative process such as incrementing a number by one for many times (1+1+1+1+1+1.....)

For more information, refer to “3. Control Structure”.

2.8 Understanding Hash

A hash is also known as associative array and has the concepts of “keys” and “values.” While an array uses a value as an index, a hash is an array that uses a string as a suffix. Unlike generic arrays, it does not have any order among their elements (that is, arrays.)

Any variable existing within a hash is called a hash variable and the name of this variable can contain alpha/numerical characters and underscore (_). Variables are not case-sensitive, however, hash keys are case-sensitive so “a” and “A” are distinguished from each other.

Referencing an element of a hash is different from an ordinary array with which you access an element of an array with a number (such as accessing the first or the second element of an array). With a hash, you use any string to reference an element of the hash (such as accessing an element corresponding to the string of xxxx.) xxxx is called a “key” that is used to reference an element of a hash.

For more information on how to manipulate a hash, “**9. Hash Manipulation**”.

2.9 Operators

An operator is a string or symbol used for describing basic processing instructions of an operation. It is one of the signs (such as “+” and “-“) used in four arithmetic operations.
This section provides tables listing operators used in WSC script.

2.9.1 Algebraic Operators

Algebraic operators perform four arithmetic operations in terms of mathematics and return a result.
The table below shows a list of symbols used in algebraic operations:

Symbol	Meaning	Description
a + b	Addition	Sum of a and b
a - b	Subtraction	Difference between a and b
a * b	Multiplication	Product of a and b
a / b	Division	Quotient of a and b
a % b	Surplus	Remainder of division a by b

2.9.2 Unary Operators

A unary operator is an operator that only takes an operator and a value for calculation.
The table below describes symbols used in unary operation:

Symbol	Meaning
+ a	Positive value
- a	Negative value
! a	Change a to one if it is zero, otherwise change it to one
~a	Take complement of each bit of a (applies only to INTEGER type)
a++	Add 1 to a
a--	Subtract 1 from a

There is one thing to note in regard to the use of [++]; a++ and ++a are different operations and you may get different results depending on the situation. Take “b=a++” for an example; the “=” sign plays a role of substitution and “a++” means “to add one,” however, because [++] comes after **a**, this operation means “to add one after substituting **b** to **a**.” On the contrary, in the operation of “b=++a”, [++] comes before **a**; therefore, this operation means “to add one and then substitute **a** to **b**.” From these examples, you should know that the position of ++ does matter in operations.
Please note that it is the same for operations including [--].

2.9.3 Bit Operators

A bit operator is an operator that performs calculations on the basis of variable bits.
If you are reading this manual from the beginning, you may remember that everything inside computers is treated as numerical values of 0's and 1's. This section provides a little more information about binary notation mentioned in "2.4 Types" to help you understand the bit operator.

Binary Notation	Decimal Notation	Hexadecimal Notation
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

Please look at the above table first. Note that the digit position of a binary number carries where it equals to 2 in a decimal number.
That is, in the operation of binary numbers, 1+1 will be 10.
The hexadecimal notation consisting of a bunch of 4-bit-data is provided for the readability of binary notation.

The table below describes bit operators used in WSC script.
As far as WSC script is concerned, these operators only work with numbers of INTEGER type.

Symbol	Meaning	Description
a & b	AND Operation	Takes bits that are set both on a and b
a | b	OR Operation	Takes bits that are set on either a or b
a ^ b	Exclusive OR Operation	Takes bits that are set only on either of a or b, and not on both

2.9.4 Bit OR Operation

A bit OR operation performs OR operation on the basis of bits.

```
a = a | b
```

Taking the above statement for example, it performs a bit OR operation and substitutes the result into a.
The arithmetic processing by the bit is performed in the following manner:

a	b	a | b
0	0	0
0	1	1
1	0	1
1	1	1

As the above table shows, if one of the bits is turned on (that is, 1), the corresponding bit becomes 1.

2.9.5 Shift Operators

A shift operator is an operator that moves (shift) bit strings to either side of your choice.
In WSC script, the only supported data is of INTEGER type.
There are two types of shift process on moving a bit string to the right direction: arithmetic shift and logical shift. An arithmetic shift is to move the bit string taking the top most sign into consideration (that is, to move – sign to the right direction) and a logical shift is to logically move the bit string to the right direction. With regard to the shift operation, ProDigio WSC script only supports *logical shift*.

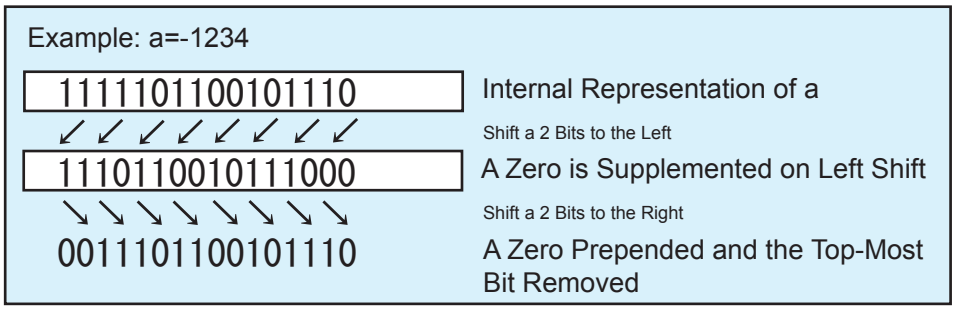
The following table shows the list of symbols used in the shift operation:

Symbol	Meaning	Description
a<<b	Left Shift	Shift a's bit b bit(s) to the left
a>>b	Right Shift	Shift a's bit b bit(s) to the right

Specific examples of shift operations are as follows:

a << 2: Shift a 2 bits to the left
a >> 2: Shift a 2 bits to the right

The following figure illustrates the above example:


Example: a=-1234
1111101100101110
Internal Representation of a
Shift a 2 Bits to the Left
11101100101110000
A Zero is Supplemented on Left Shift
Shift a 2 Bits to the Right
0011101100101110
A Zero Prepended and the Top-Most Bit Removed

2.9.6 Comparison Operators

A comparison operator is an operator used in conditional expressions where you compare two values, expressions, or strings and returns either 1 or 0 depending on the comparison result.
For string comparison, their size is evaluated based on their order in the ASCII code list.
The following table shows the list of symbols used in the shift operation:

Symbol	Meaning	Description
a==b	Equal	Returns 1 if a is equal to b
a!=b	Not Equal	Returns 1 if a is not equal to b
a<b	Less	Returns 1 if a is less than b
a>b	More	Returns 1 if a is more than b
a<=b	Less or Equal	Returns 1 if a is less than or equal to b
a>=b	More or Equal	Returns 1 if a is more than or equal to b

It is very useful to use comparison operators in conjunction with conditional expressions such as an **if** statement. Details of conditional expressions are provided in the next chapter.

2.9.7 Logical Operators

A logical operator is an operator that returns either “true” or “false” and is use when combining more than two factors to get a result out of them.
You may have not noticed it, however, this logical operation is performed frequently when using computers.
You probably have done research on the Internet using a search engine and entering keywords, and possibly using spaces to refine your search results. This is an example of where logical operation is employed. You can make up various combinations of search criteria. Say you want to search for pages containing a keyword “A”; you could search for pages that contain not only A but also keyword B or that include either A or B.

Here is an example: Number **a** has a fixed value of 0-100. When the user enters in a value for the number **b**, the script works out the absolute value of the difference between value **a** and b and runs a process based on the following criteria:

- “False” if the value is more than 10
- “Neither” if the value is less than 10
- “True” if value is equal to the value **a**

To create a branch process that meets the above criteria, you will need to provide a condition that the difference between **a** and **b** satisfies all of the following three events in order to determine if the difference between two values is less than 10 (that is, “Neither”) and proceed on to the appropriate process:

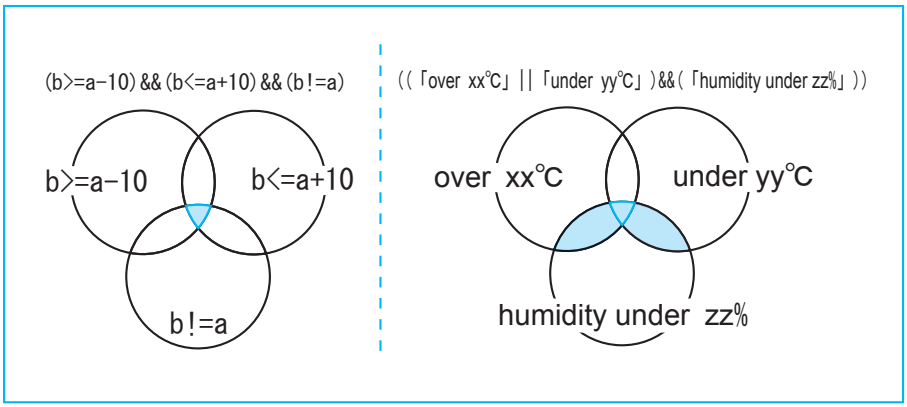
- Event 1: **b** is larger than the result of **a** minus 10 (b>=a-10)
- Event 2: **b** is smaller than the result of **a** plus 10 (b<=a+10)
- Event 3: **b** is not equal to **a** (b!=a)

Combining these criteria is referred to as a “logical operation.”
With ProDigio WSC script, you can use both AND operator (a and b) and OR operator (**a** or **b**) to determine if an event is “true” or “false.”

The following is the list of symbols used in logical operations:

Symbol	Meaning	Description
a&&b	AND	Returns 1 if neither of values, a nor b, is 0
a||b	OR	Returns 1 if either a or b is not 0

An example of expression stating the condition “meeting all of the following three events” will look like the following: (b>=a-10)&&(b<=a+10)&&(b!=a).
To be more explicit on the use of ProDigio, you can write a condition of “(a or b) and c”, for instance, “issue a warning when temperature is higher than xx°C or lower than yy °C, and humidity is lower than zz%” in the following way: ((“over xx°C”||”under yy°C”)&&(“humidity under zz%”)).



2.9.8 Priority of Operators

You can combine multiple operators within an expression to perform multiple processes, however, it is important that you know the order in which those operators are performed.
You may end up getting unexpected results if you do not understand the exact processing order of operators.
Please note that every operator except for assignment operator is performed in left to right order and an assignment operator is performed from right to left.

[Priority of Operators]

High↑
- ()
- [ ]
- ! ~ + - ++ -- (Unary Operators)
- * / %
- + -
- << >>
- < > <= >=
- == !=
- &
- |
- ^
- &&
- ||
- =

Low↓

2.10 Expressions

An expression is a group of variable-operator combinations.
You can also enclose an expression that stands for a special meaning in “( )” as shown in the following example:

[Sample Expression]

```
str = ((1+5)*3);
```

Operators represent variable calculation or comparison criteria in a form of expression and all expressions return a value as a result. Most operators are used for operations between data of the same type, either INTEGER or FLOAT.
The operations available for STRING type data are addition and comparison only.

[Example of String Operation]

```
var  str[5];
str = "abc" - "edf" ;    //  error
```

When operating data of STRING and INTEGER types, INTEGER type data will be converted into STRING.

[Example of String and Integer Data Operation]

```
var  str[5];
str = "aBc" + 1;                  //aBc1
```

Please keep in mind that string operation is case-sensitive.

2.10.1 Variable Type Conversion

Type conversion is a process of changing a value of X type data into a value of Y type data.
In the preceding example, you can see that a value of INTEGER type is converted into a value of STRING type through the expression with an operation.

When you do substitution or operation of values between variables of different types, the value type of the source variable is automatically changed to that of the target variable. The precision of the variable is aligned to that of the target variable, if it has shorter precision than the source variable.

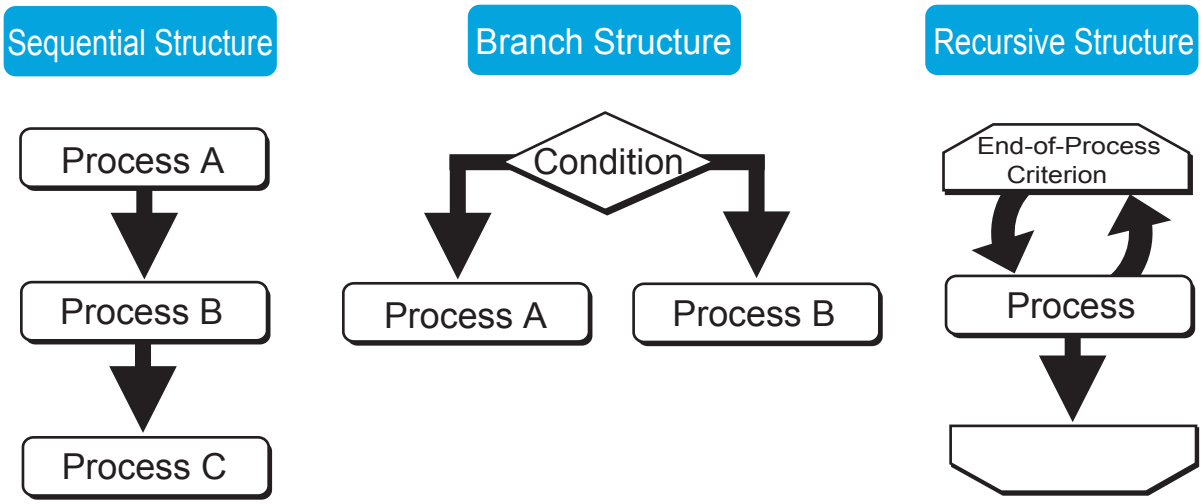
Type conversion is useful when you want to change the type of data stored in a variable just for the time being.

An error will be displayed and script execution will terminate, if the script could not convert a variable into the type of target variable or when it is not possible to conduct an operation among different types of data.

3. Control Structure

When programming, you should probably write commands from top to bottom in the order you want the program to perform.
For example, assume that there is a program that has finished with the first block and is directing the computer to move on to the second block. What if you want the program to do the same process in the first block after completing the series of processes included in the second block? Do you still need to write the same code beneath the second block again? The answer to this question is “No.” It is inefficient and moreover the size of the program would get larger for no real reason - there are no good points.

A control structure is not just a simple structure that executes statements in top-to-bottom order (sequential execution) but manages program flow by changing the executing statements based on the criteria and re-executing the same statements more than once.
This plays a very important role in determining the program structure and this whole process flow is called “statement.” Flowcharts used in this section are a common method for presenting the graphical image of statements, but there are other methods such as the one using pseudo language.



Sequential Structure	:	Format in which the processes are performed in the order they appear
Branch Structure	:	Format in which processing is branched according to conditions(if•else statement)
Recursive Structure	:	Format in which processing is performed over and over(while•do..while•for statement)

All control structures must be enclosed with { } to be divided into blocks.

This chapter provides basic information about control statements building up these control structures.

3.1 if statement

An **if** statement is used for conditional execution.
The purpose of this statement is to execute process A only if a given condition is met. This statement is called “conditional expression” and described as follows:

[**if** Statement Syntax]

```
if(expression){
        statement;
}
```

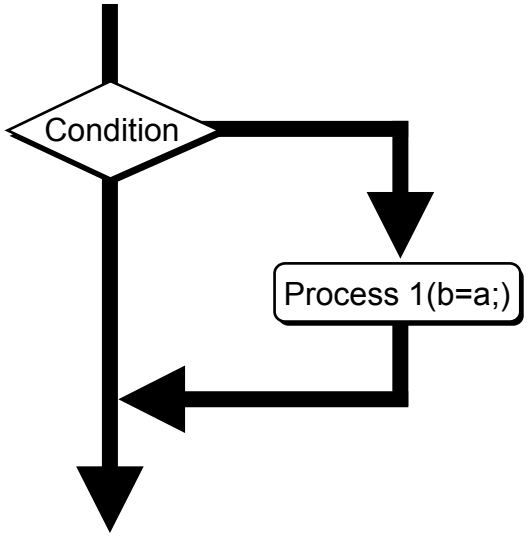
By writing the code using the above syntax, the program executes the statement enclosed in the **if** statement only when the expression returns zero. The statement will not also be executed, if the expression returns a string of “ .“

The following is an example of an **if** statement:

[Example of an **if** Statement]

```
if(a > b) {
        b = a;
}
```

In this case, you should be able to follow that if a is greater than b, **if** statement is executed, directing the program to substitute a into b.



3.2 else statement

An **else** statement specifies the process to be executed when the condition in the **if** statement is not met.
In other words, in case the condition in the **if** statement is met and the statement is executed, the statement in the **else** clause will not be executed.

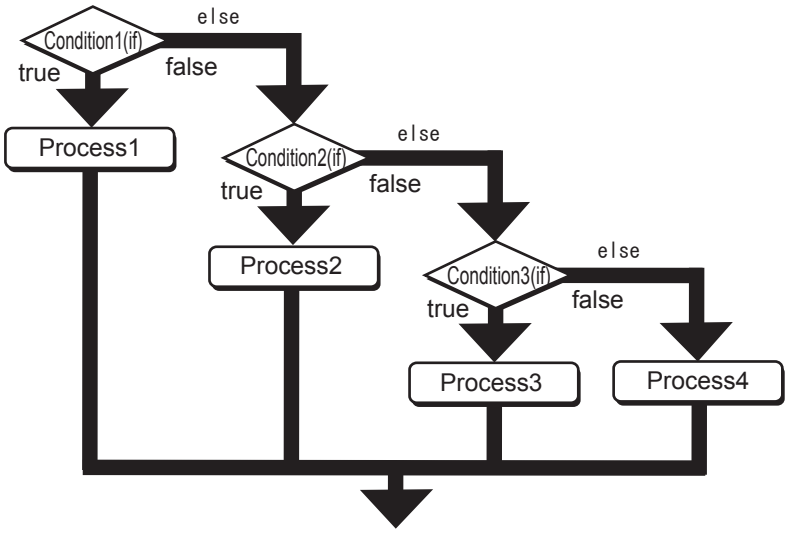
[Example of **else** Statement]

```
if(a > b){
        b = a;
}else{
        a = b;
}
```

The preceding sample tells the program to substitute a into b on condition that a is larger than b; if a is smaller than b, the condition specified in **if** statement will not be met.
The subsequent **else** statement is provided to tell the program to assign b into a if a is smaller than b.
You can also include **if** statements within **else** statements to provide more conditional expressions.

[Example of Combining **if** Statements and **else** Statements]

```
if(a == 0){
        b = 0;
}else if(b == 0){
        a = 0;
}else if(a > b){
        b = a;
}else{
        a = b;
}
```



3.3 while statement

A **while** statement repeats the execution of statement(s) as long as the given condition remains true and exits when the condition no longer holds valid.
Simply put, **while** statements can be used when you give a direction such as “iterate the operation of adding one to a variable whose initial value is one and move on to the next step when it reaches a hundred.”

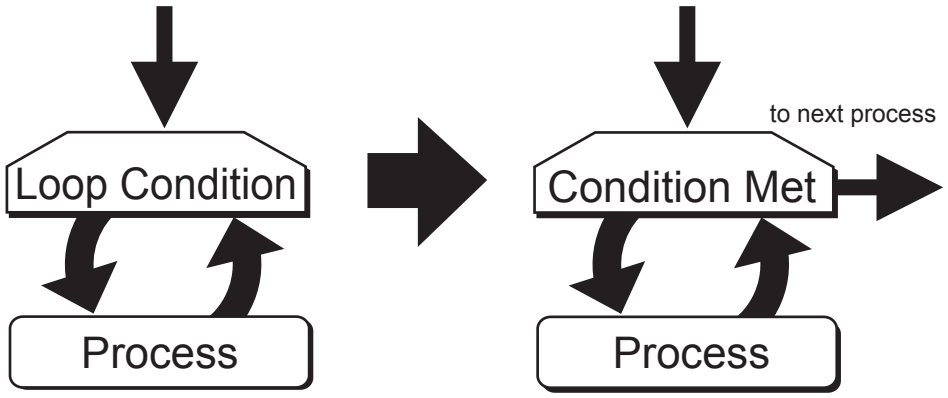
[**while** Statement Syntax]

```
while(expression) {
        statement;
}
```

In the above code, the statement will be iterated until the specified condition is met.

[Example of **while** Statement]

```
var i;

i = 1;
j = 1;
while(i <= 100){
        i = i + 1;
}
```



When Condition is Not Met

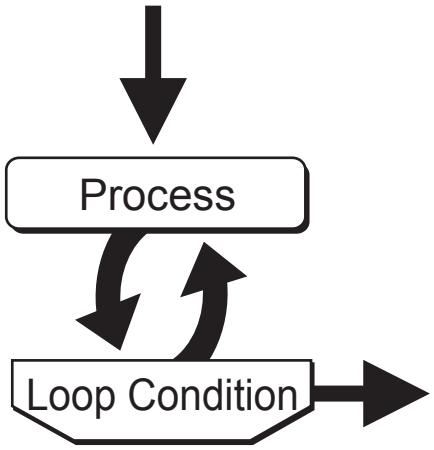
When Condition is Met

3.4 do..while statement

A **do..while** statement is similar to a **while** statement in that it repeatedly executes statement(s) as long as the conditional expression is satisfied. The difference between these statements is that a **while** statement validates the condition before executing the statement whilst a **do..while** statement validates the condition after executing the statement. In short, the validation point is different from **while** statement.
Therefore, a **do..while** statement executes the recursive process at least once even if the condition is NOT met.

[**do..while** Statement Syntax]

```
do{
        statement;
}while(expression);
```



Process Gets Executed at Least Once

3.5 for statement

A **for** statement is used to execute the recursive process only for the specified number of times. This loop statement can also be used to initialize a variable or specify the range in which a variable can change.

[**for** Statement Syntax]

```
for(expression_1; expression_2; expression_3) {
                    statement;
}
```

expression_1 is executed before entering the loop and will be executed recursively while the condition, *expression_2*, is met.
expression_3 is executed at the end of each loop cycle.

[Example of **for** Statement]

```
for(i = 1; i <= 10; i++){
                    j = j + i;
}
```

In the above example, expression_1 instructs to substitute the value of one into i and expression_2 specifies to repeat the process while i is equal to or less than 10. Expression_3 tells to add one to the value of i and the value of i will be incremented by one when it goes through a cycle of the recursive process, expression_1 - expression_3.
The value of i eventually reaches 10 by going through the process 10 times. When the value of i reaches 10, this recursive (loop) process will end.
It is possible to write a **for** statement without expressions_1, 2 or 3. However, you need to be careful with the above example; if you do not provide expression_2 , the process will enter an infinite loop.

3.6 break statement

Use a **break** statement when you want to break out of a current loop.
If you specify a value with a **break** statement, the process will break out of the loop process at a specified number. The process of the function will be terminated if you specify a value that is larger than the number of loop processes existing in the function.
If no value is specified with a **break** statement, it will have the same effect as specifying **break 0**;

[**break** Statement Syntax]

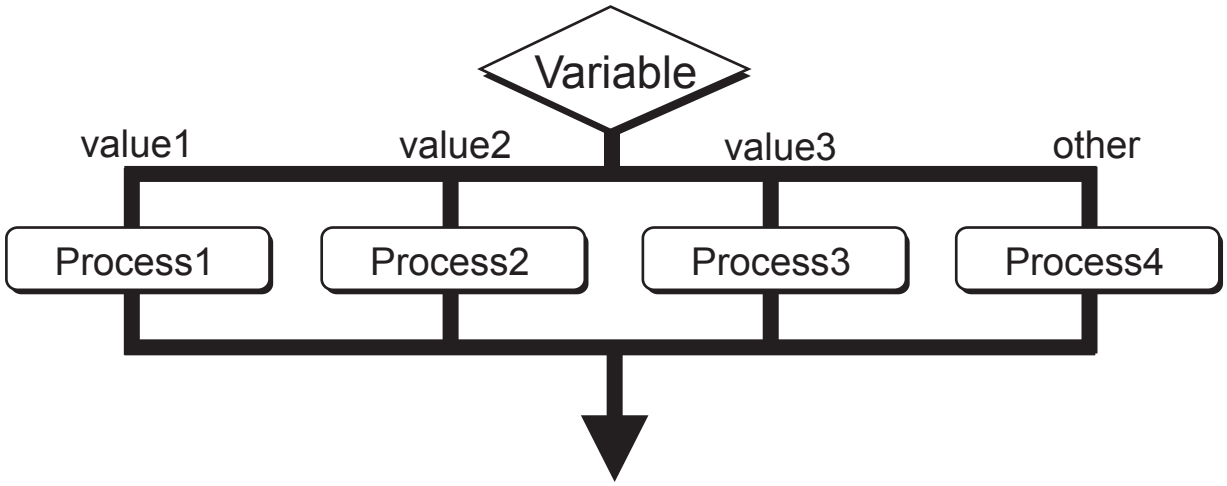
```
while (expression_1){
        while (expression_2){
                while (expression_3){
                        break; //break, break1, and break3 force the
                                //process to get out of expression_3,
                      // expression_2, and expression_1 respectively.
                }
        }
}
```

3.7 continue statement

A **continue** statement is used when you want to make the process go back to a loop control statement.
With a **continue** statement, you can cancel the remaining statements in the current loop process and jump on to the next recursive process.

3.8 switch statement

A **switch** statement is a conditional branching statement that selects the best matching statement among several statements within a block based on the condition and executes the appropriate statement. You can use **if** statements that were mentioned earlier in this chapter to perform a similar process. In some cases, however, it is more effective to use a **switch** statement rather than **if** statements.



[**switch** Statement Syntax]

```
switch(expression)
{
case value_1:
      statement_1;
      break;
case value_2:
      statement_2;
      break;
case value_3:
      statement_3;
      break;
default:
      statement_4;
      break;
}
```

Notice that you can use a string variable in a **switch** statement. In this case, you can use strings for the value of **case** statements.

[Example of Specifying String Variable as a Variable of Switch Statement]

```
var   a;

switch(a)
{
case “abc”:
  statement_1;
  break;
case “def”:
  statement_2;
  break;
default:
  statement_3;
  break;
}
```

The process proceeds until meeting a **break** statement or the end of block if a matching expression and value is found from one of the **case** statements.

A **default** statement will be executed if no matching case statement exists.

3.9 Comments

It is recommended that you add some comments at the end of a statement or a block to help you or others see what the statement or block is for.

Write comments to point out the purpose of the script or to explain any other intentions.

[Example of Comments]

```
// e-mail delivery process
```

OR

```
/* e-mail delivery process*/
```

4. Using Escape Characters

An escape character is used to represent a line break, space, and so on.
Brackets are drawn around the output to the terminal console.

4.1 \n \r

[Sample Code]

```
puts("\nABC\rDEF\n") ;
```

[Output To Terminal]

```
ABC
DEF
```

"\r\n" indicates a line break (\r=CR,\n=LF.)

4.2 \t \b

[Sample Code]

```
puts("\tABC  \DEFn") ;
puts("\tABC  \bDEF\n") ;
```

[Output To Terminal]

```
ABC  DEF
 ABCDEF
```

4.3 \\ \" %%

[Sample Code]

```
puts("\\ \" 25%%\n") ;
```

[Output To Terminal]

```
\"25
```

4.4 \x

[Sample Code]

```
puts("\x50\n") ;
```

Display the character corresponding to the ASCII code 0x50 (that is, "P")

[Output To Terminal]

```
P
```

4.5 \0

[Sample Code]

```
puts("\061\n") ;        Display the character corresponding to the ASCII code
                        61 (in octal notation)  =0x31=”1”
```

[Output To Terminal]

```
1
```

5. Using Functions

A function in the world of programming is different from that in mathematics. Here, it is a series of statements that receive data called arguments[*1], execute the specified set of processes, and return the result.

For instance, an order stating to “get the values of x and y, and return the value resulting from the addition of these two values” is a function and the name of this function is “a.” Then, assume that “by providing function **a** with two arguments of 4 and 5, it will return a value of 9. Some functions are capable of displaying not only numerical values but also strings on to the console.

The following is the syntax of a function:

```
Function function_name()
{

}
```

When calling a function, use the following line of code

```
Function_name();
```

WSC script always requires a **main function**.
When a script is invoked, it starts executing the **main** function[*2] first.

[*1] Argument
An argument is information that a function requires to perform its tasks and each argument possesses a type and a name.
When providing more than one argument, separate them by commas (,).

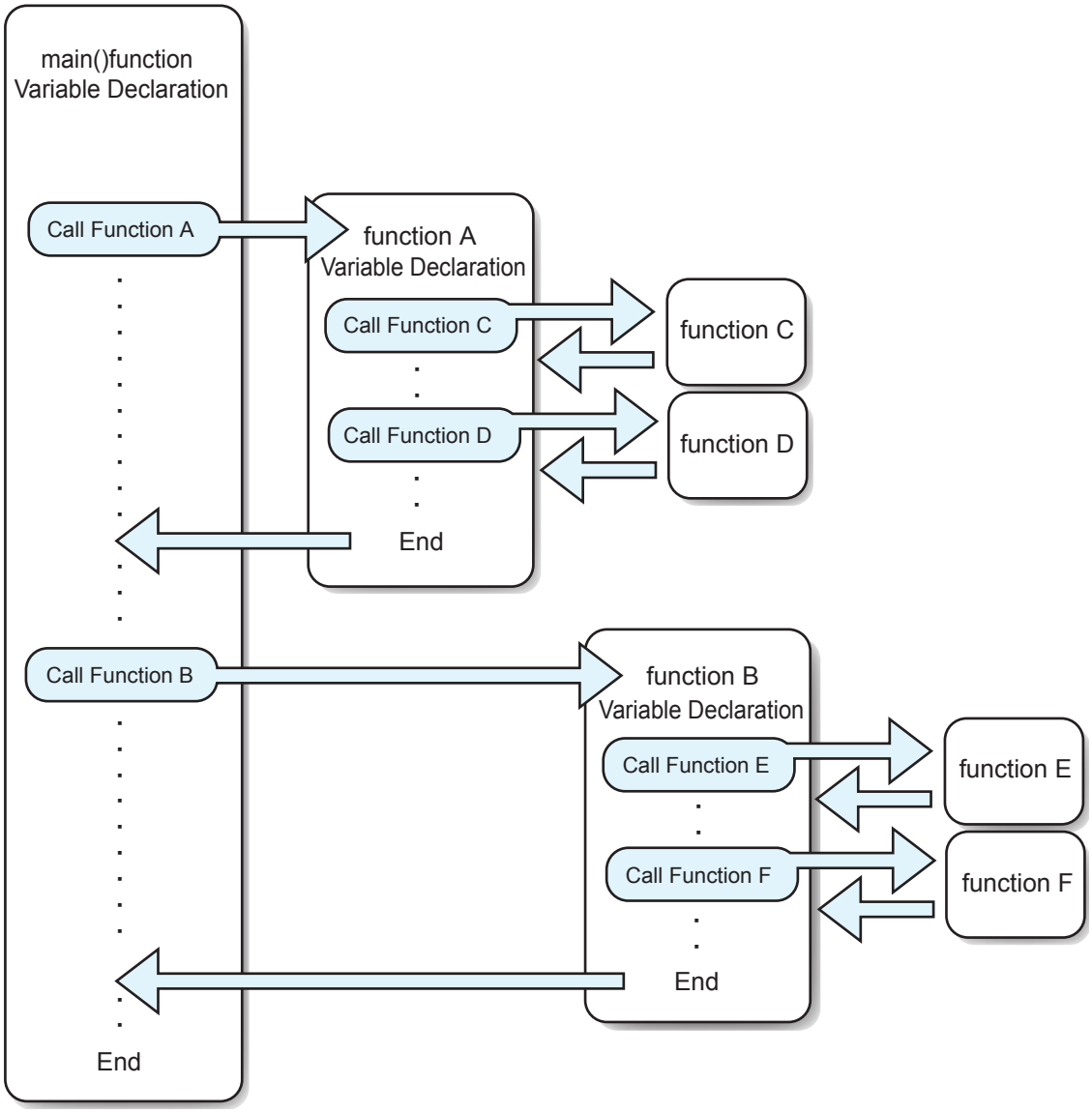
[*2] main function
A main function is the first function to be executed when the script is invoked.
Make sure to provide a main function because script processing proceeds based on the statements written in a main function.
A main function is the starting point for all the processes and subsequent functions are called out in relays from there.

Keep the following points in mind when writing a function:

1. A function name is not case-sensitive.
2. Where you write a function does not matter, it can be placed prior to or subsequent from the function that it is being called from.
3. A function generally returns a value of zero, but you can configure it to return a different value by inserting a line of “return Return_Value;”.
4. Prepending an & (ampersand) in front of an argument makes its return value passed by ref.

   Pass By Ref is a method of passing the location (address) of the argument internally to the function called.

Note: For more code samples, refer to “5.1 Basics of Using Functions.”



The figure above is an illustration of the relationship between functions.

This figure shows how a program and functions are related to each other. Any complex processes can be achieved by combining functions. For example, you can work out a complex process using three functions to get a final result; function **a** calls function **b** to get a return value, and the return value is passed on to function to get the final result.

By combing functions in a thoughtful manner as in the preceding example, you can achieve the most efficient process flow.

5.1 Basics of Using Functions

This section provides a typical example of using functions.
If you run the following sample script, you will get the result of rtn1=10 and rtn2=10.

```
// sample function
// function that returns a result of simple addition

// function test1
Function test1(a1, a2)
{
	return a1+a2 ;
}

// function main
Function main()
{
	var rtn1 ;
	var rtn2 ;

	rtn1 = test1(2,7) ;
	test2(rtn2) ;
	puts(“rtn1(%rtn1%) rtn2(%rtn2%)\n”) ;
}

// function TEST2
Function TEST2(&b1)
{
	b1 = b1 + 1 ;
}
```

5.2 Retrieving Arguments when CGI is in Use

Arguments of a function are stored in an **arg[ ]** array[*3] when CGI is being used.
The number of the arguments stored in this array can be retrieved by using **gethashcount** function[*4].

[Example: test1.wcr Called from Browser with the Following Arguments]

Arguments of CGI: a1=4 and a2=6

http://192.168.1.1/cgi-bin/test1.wcr?a1=4&a2=6
Arguments

[Contents of **test1.wcr**]

```
// how to extract arguments when CGI is in use
Function main()
{
    var arg1;                          // value for argument1
    var arg2;                          // value for argument2
    var add1;                          // result of addition 1
    var add2;                          // result of addition 2

    arg1 = mid(arg[0],3);          // 1st argument
    arg2 = mid(arg[1],3);          // 2nd argument
    add1 = arg1 + arg2;            // add strings (add1 equals to “46”)
    add2 = int(arg1) + int(arg2);  // add numerical values (add2 equals to 10)
}
```

[Description]
The arguments stored in an **arg[ ]** variable are “strings.” In this example, **arg[0]** is “a1=4” and **arg[1]** is “a2=6.” In the process of obtaining these arguments, this function receives 4 and 6 respectively in arg1 variable and arg2 variable, excluding “a1=” and “a2=.”
As a result, the **add1** variable handles the addition of strings and the result will be a string of “46.” On the contrary, because the **add2** variable converts strings into numerical values before addition, the result will be the numerical value of 10.

[*3] arg[ ] array
See “5.3 arg[ ] Array”.

[*4] gethashcount function
See “5.4 gethashcount Function”

5.3 arg[ ] Array

An **arg[ ]** array is a place where arguments retrieved by operating WSC script as CGI are stored. If you get an argument from the browser by CGI and want to use it in the script, call the corresponding value stored in the arg[ ] array.

You can use the value extracted from an **arg[ ]** array for various purposes including processing it through the use of expressions and getting values stored in a specific range through the use of functions.
The sample script in the previous section gets arguments of “a1=4” and “a2=6” from the browser, and extracts the last letter of those arguments by using a **mid()** function.
Those extracted values are processed through expressions and WSC script uses the argument information that it got from the browser.

The **arg[ ]** array is provided exclusively for retrieving arguments from the browser.
Therefore, do not declare it with **var**.

5.4 gethashcount Function

The **gethashcount** function is a function used to get the number of items in an associative array (hash.)
If you know the number of items to be stored in an **arg[ ]** in advance, you do not need to use **gethashcount()**. However, if the number of arguments to be passed from the browser is unknown, you have to get the number of items in the hash. In such a case, use this function.

6. Working with CGI and SSI

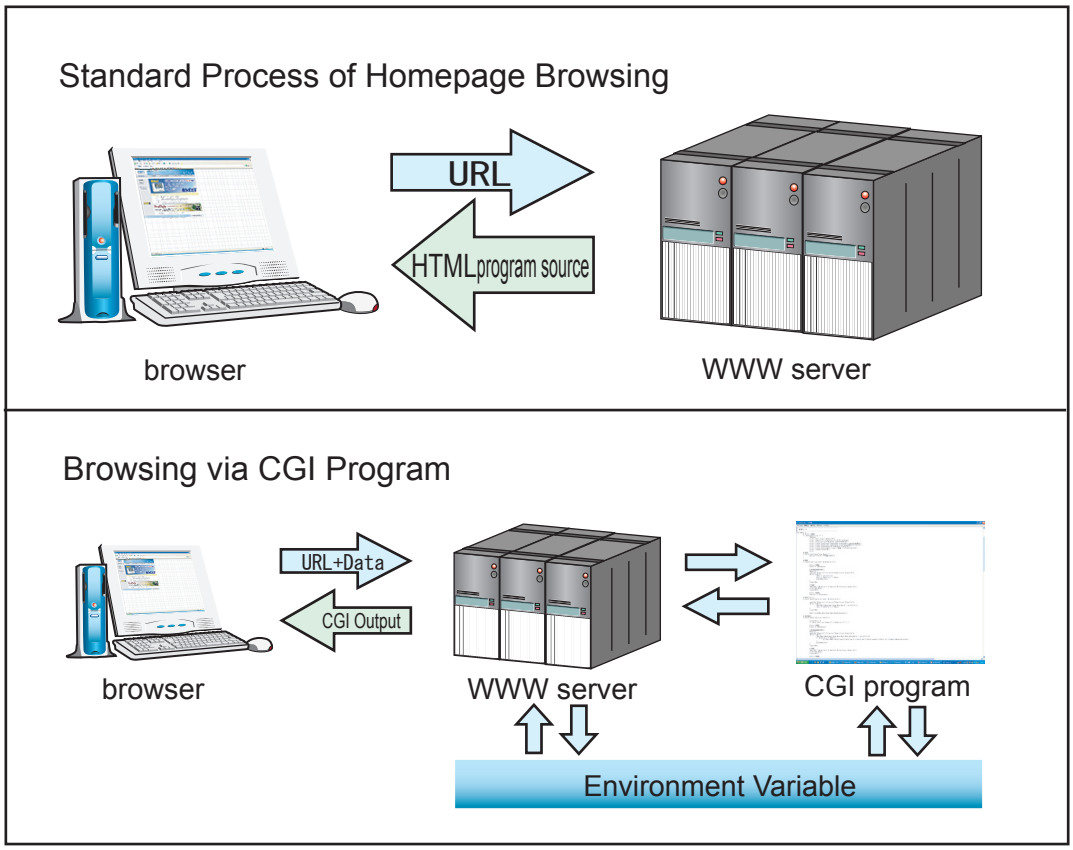
6.1. What is CGI?

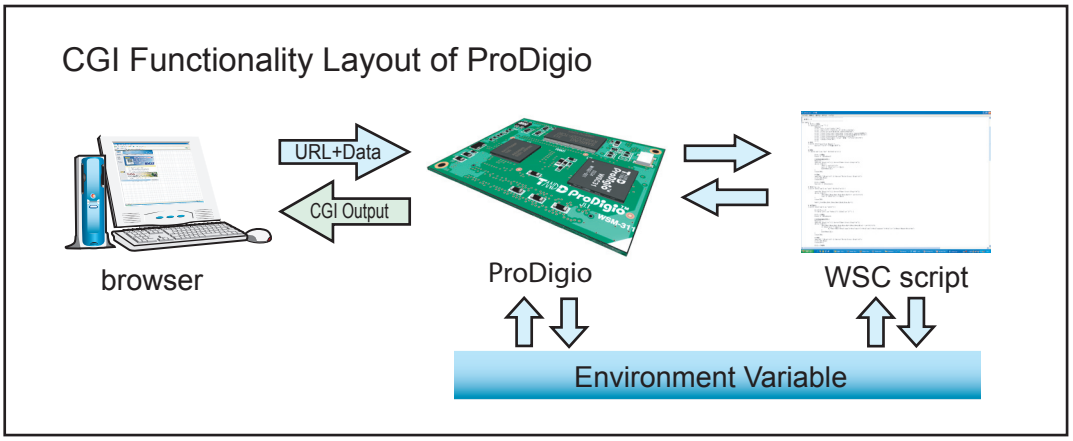
The Common Gateway Interface or CGI is a mechanism (or an interface) that executes scripts on the server rather than on clients.
Things you often run into on the Internet such as bulletin boards, access counters, shopping carts, and many forms that allow users to submit a post or question from within a page are all implemented by CGI. There are variety of programming languages available for CGI such as C, Perl, or Ruby language. However, the availability of language varies depending on the server specification.

CGI programs enable real-time interactive communication and communication history logs while HTML offers only one-way, outgoing, communication.

A Webpage is a reproduction of a program source written in compliance with HTML language specifications by loading it from the Internet into a browser on your computer. Thus, you must be careful when creating a page because a page can be rendered differently or not even rendered, depending on versions or types of browsers.

ProDigio runs WSC scripts as CGI, allowing you to control ProDigio by using the Internet browser from a remote site, and to set various devices or retrieve and view data via ProDigio.





6.2. What is SSI?

The Server Side Include or SSI is a mechanism that processes an html file before passing the content of the file on to the client who requested the *.html to the server, unlike normal web browsing where the server passes the content of the file as it is. This process does not modify the file on the server, but only processes the data that the server returns to the client. Here, “processing” means that you write commands into an HTML or text-based file in advance and the server rewrites that part of the file in compliance with the commands when the file is requested, then returns it to the requesting client.

Both CGI and SSI programs work on ProDigio because they work on the server-side. By default, ProDigio provides CGI and SSI features. To utilize them, you can write CGI or SSI programs with a WSC script.
For details about CGI and SSI, see retail CGI and SSI reference manuals, as there is too much information to cover in this documentation.

There are two different ways to run a script from your browser:

Method 1: running WSC script by form (See 6.3.)
Method 2: running WSC script by SSI (See 6.4.)

6.3. Running Script by Form

Here is an example of entering three numbers from a form on index.html and displaying the sum of them on the browser.

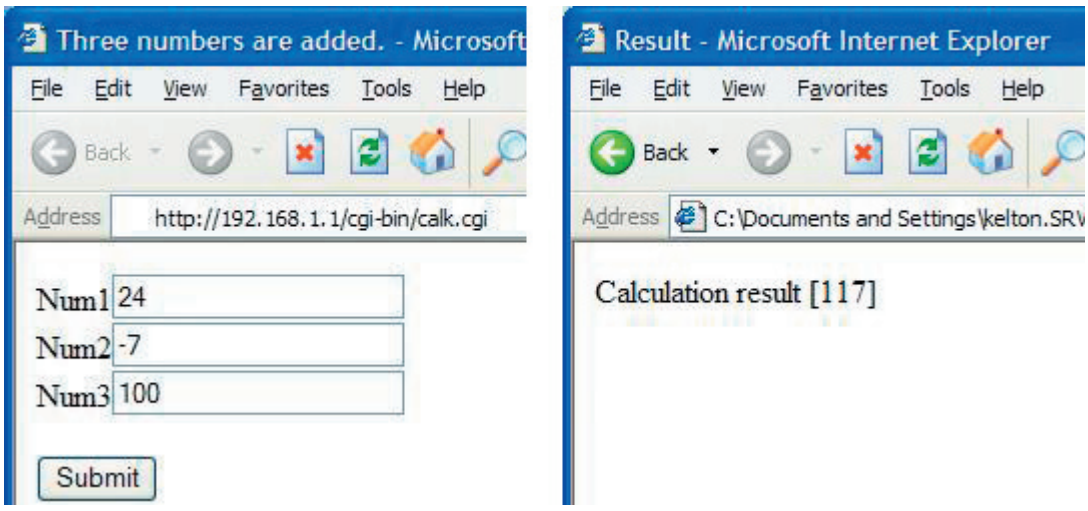
[Contents of **index.html**]

```
<HTML>
<HEAD>
<TITLE>Addition of Three Values</TITLE>
</HEAD>
<BODY>
<FORM  ACTION=”add1.wcr”>
Value 1
<INPUT TYPE=”text” NAME=”param1”><BR>
Value 2
<INPUT TYPE=”text” NAME=”param2”><BR>
Value 3
<INPUT TYPE=”text” NAME=”param3”><BR>
<INPUT TYPE=”submit” VALUE=”Submit”><BR>
</FORM><BR>
</BODY>
</HTML>
```

[Contents of **add1.wcr**]

```
// display the result of adding three values on the browser

Function main()
{
 var data1 ;
 var data2 ;
 var data3 ;
 var val ;

 // assign arguments to internal variables
 data1 = mid(arg[0],7) ;
 data2 = mid(arg[1],7) ;
 data3 = mid(arg[2],7) ;

 val = int(data1) + int(data2) + int(data3) ;

 // display the result on browser
 webprint ("<html>\r\n");
 webprint ("<html><head><title>Result of Calculation</title><head>
           \r\n");
 webprint ("<body>\r\n");
 webprint ("<br>\r\n");
 webprint ("The result of calculation is %val%<br>\r\n");
 webprint ("</body>\r\n");
 webprint ("</html>\r\n");
}
```

[Output of Sample Script]



When running WSC script as CGI, the maximum length of each argument passed from the browser is 255 characters long. However, note that arguments cannot contain any spaces.

6.4. Running Script by SSI

Here is an example of starting a script that retrieves and shows the current time from ProDigio when opening index.html. This invokes the script to return a time string on opening the index. html page and writes the output to the file. This example assumes that **nowTime.wcr** file is located in **C:\www\cgi-bin** directory.

[Contents of **index.html**]

```
<HTML>
<HEAD>
<TITLE>Displaying the Current Time</TITLE>
</HEAD>
<BODY>
The current time is
<!-- #CGI_EXEC {\cgi-bin\nowTime.wcr} -->
.
<BR>
</BODY>
</HTML>
```

[Contents of **nowTime.wcr**]

```
Function main()
{
    var nowTime ;
    var time1 ;
    var time2 ;
    var str ;

    nowTime = getrtcTime() ;       // get current time
    time1 = getdatestring("yyyy MM dd", nowTime) ;
                                   // convert from date to string
    time2 = gettimestring("HH mm ss", nowTime) ;// convert from time to string
    str = time1 + " " + time2 ;

    webprint("%str%") ;
}
```

Considerations on SSI Argument:

When invoking script using SSI, arguments are passed in the following way:

<!-- #CGI_EXEC {\cgi-bin\nowTime.wcr a=2 b=5} -->

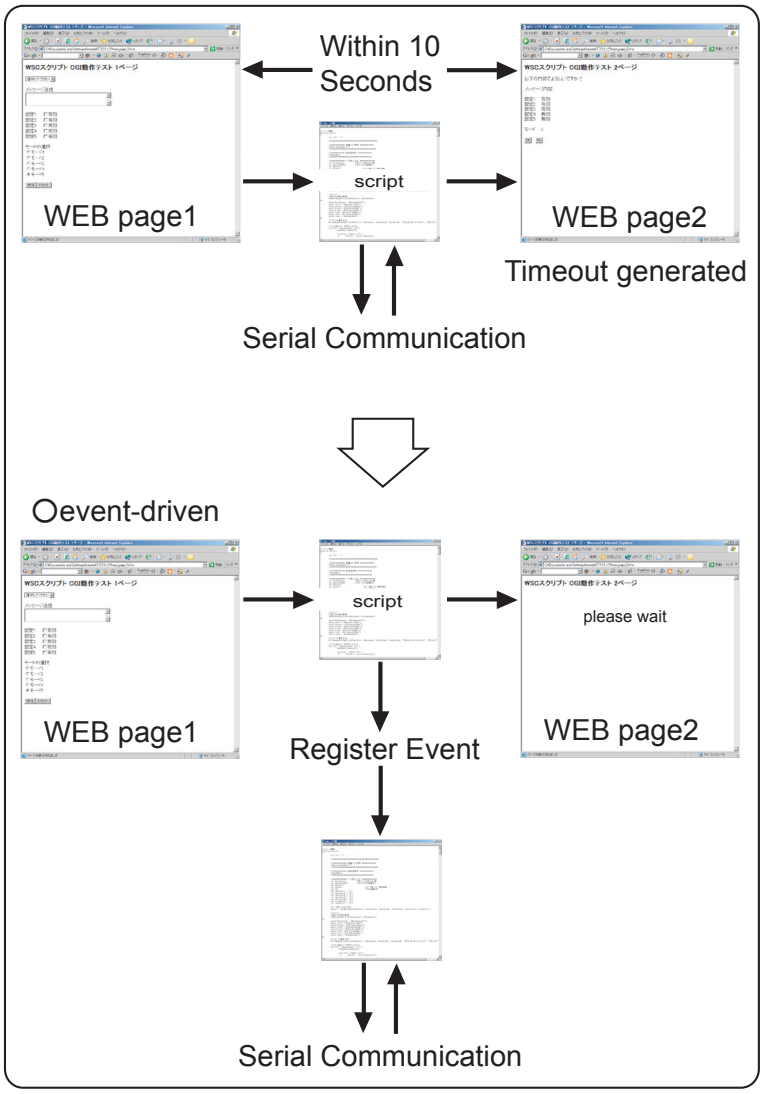
In this example, the script receives strings of "a=1" and "b=2".
Use of =, { }, ", and space for an argument is not allowed.

6.5. CGI and SSI Timeouts

Use of CGI or SSI in the script limits the lifetime of the script to 10 seconds. When you design and create a script, remember if a script runs longer than its lifetime, then the WSC module is restarted automatically.
If your design is inappropriate, for example, the module which user A is using to try to view data in his browser might be restarted automatically against his will when user B starts a different process in her browser.
It is also important to note that scripts cannot run concurrently (that is, parallel execution is not supported.)

6.5.1. Example of Serial Communication

If you perform serial communication from within Web page 1 to load the result into Web page 2, waiting for the end of serial communication before rendering web page 2 leads to a timeout.
In this case, prepare an event-driven script and set the script startup to trigger up on receiving data to “SIO.”



6.5.2. Notes on Occurrence of Other Events

You must consider that another event may occur even if an event-driven script has been provided. In the preceding serial communication example, for instance, it is possible that when user A is executing an event-driven communication and user B sends a communication request, the communication of user A may be interrupted or he may receive unexpected data.
This is because events are started in the order of event request.
That means some communication commands sent by user B may interrupt a sequence of the communication of user A.
To reduce the possibility of this issue, you need an exclusive control such as creating a common file inside ProDigio to check before starting a script.

6.6. Linking from HTML File to Drive Other Than C Drive

When you have configured HTTP server **test.com** on ProDigio and want to specify a link to **B:\aaa.html** in **C:\www\index.html**, you can use the following code.
All the below examples behave in the same way regardless of their different representations.

```
<a href=http://test.com\B\aaa.html>Test Link</a>
<a href=http://test.com\b\aaa.html>Test Link</a>
<a href=http://test.com/B/aaa.html>Test Link</a>
```

Note: For details on HTML representation described in this section, see retail HTML reference manuals.

7. Time Functions

Time functions are used to retrieve the current time through a program.
This section shows an example of setting a date/time from standard input.
Note that rebooting your development kit after setting the time resets the time to the default date/time (Jan 1, 2004 00:00:00.)

[Example Code of Retrieving the Current Time]

```
Function main()
{     var nowTime ;
      var time1 ;
      var time2 ;
      var str ;
      var input ;
      var setTime[ ] ;
      var rtn ;
      nowTime = getrtcTime() ;       // get current time
      time1=getdatestring("yyyy/MM/dd", nowTime) ;// convert from date to string
      time2 = gettimestring("HH:mm:ss", nowTime) ;// convert from time to string
      str = time1 + " " + time2 ;
      puts("%str%\n") ;        // display current time
                               // input date/time
                               // date/time format is YYYYMMDDhhmmss
      input = gets(14) ;
      if(len(input) != 14)
      {
            error("Number of characters incorrect\n") ;
            return ;
      }
      setTime["year"] = mid(input, 0, 4) ;
      setTime["month"] = mid(input, 4, 2) ;
      setTime["day"] = mid(input, 6, 2) ;
      setTime["hour"] = mid(input, 8, 2) ;
      setTime["minute"] = mid(input, 10, 2) ;
      setTime["second"] = mid(input, 12, 2) ;
      // set time
      rtn = setrtctime(setTime) ;
      if(rtn < 0)
      {
            // error handling
            error("Error setting time") ;
      }
}
```

8. File Manipulation

This chapter describes a productive way of manipulating files when processing a large amount of data in the scripts.

The basics of this file manipulation are as follows:

- - - Basics of File Manipulation - - -

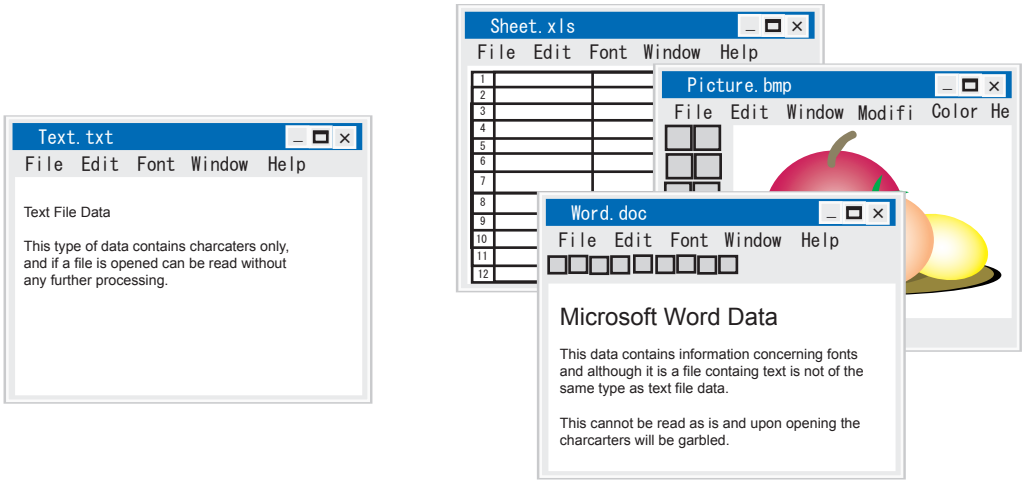
- Save data.
- Read the saved data.
- Input the data using files.
- Output the data using files.

As explained in “2.4.4 Array“ and “2.7 Understanding Array”, you can use an “array” as a means to programmatically process a large amount of data in one go. With arrays, you can create script that is capable of loading many values and performing a variety of tasks, however, values stored in an array are cleared from memory when the process of the function finishes.
You can save values or refer to the saved data through the use of files.

This kind of file can be used for various purposes. For example, you can use data generated by a script in another script or to display data with an image file obtained through ProDigio.

8.1 File Types

Types of files that computers can process fall into two categories.
One is “text file,” which is created with an editor with which you can read characters and symbols written in the file.
The other is a “binary file,” a file type in which computers can directly read and interpret. Any files other than text file belong to this category.


Text.txt
File Edit Font Window Help
Text File Data
This type of data contains charcaters only, and if a file is opened can be read without any further processing.
Sheet.xls
File Edit Font Window Help
Picture.bmp
File Edit Window Modifi Color He
Word.doc
File Edit Font Window Help
Microsoft Word Data
This data contains information concerning fonts and although it is a file containg text is not of the same type as text file data.
This cannot be read as is and upon opening the charcarters will be garbled.

text data　　binary data

Simply put, script you created is a text file, because you can read it as is. On the other hand, computer’s application software is a binary file, because you cannot open the file and read its content as characters (even if you could open the file, the content of the file would be garbled.)

8.2 Listing All Files in a Directory

The following script lists all file names and directory names that exist under the current directory. You can open a directory with **diropen** function. However, you must remember that if you open a directory, you need to **close** it with a close function.
Please note that if you forget to close the directory, you will not be able to access the file system and may end up having to restart ProDigio.

```
// script to list all files in a directory

Function main()
{
  var    dirNo  ;        // directory number
  var    fName;         // file name
  var    n = 0   ;

  // open directory
  dirNo = diropen("C:\\www\\cgi-bin") ;
  if(dirNo < 1)
  {
        error("Dir Open err\n") ;
        return ;
  }

  // search directory for files and store the result in a file
  while(1)
  {
        fName = dirread(dirNo) ;
        if(len(fName) == 0)
        {
              // end search upon reaching the end of directory
              break ;
        }
        else
        {
              n = n + 1 ;
              puts("%n% %fName%\n") ;
        }
  }

  // close directory
  dirclose(dirNo) ;
}
```

8.3 Input/Output Text File

The following script loads a text file, adds two lines, and then outputs its content to another file.

[Contents of the Original Text File (sample.txt)]

```
Baa, baa, black sheep,
Have you any wool?
Yes, sir, yes, sir,
Three bags full;
```

[Lines to Add]

```
One for the master,
And one for the dame
```

The following sample script shows how to add lines of text to an existing text file:

```
// sample script of the I/O text file

Function main()
{
	var	fd		;	// file descriptor
	var	buf		;	// write buffer
	var	size		;	// file size
	var	rtn		;

	// open the file
	fd = fileopen("C:\\www\\cgi-bin\\sample.txt", "r") ;
	if(fd < 0)
	{
		// open file error
		error("File Create Err") ;
		return ;
	}
	// retrieve file size
	size = getfilesize(fd) ;
	// read file
	rtn = fileread(fd, buf, size) ;
	if(rtn < 0)
	{
		// read file error
		error("File Read Err") ;
		fileclose(fd) ;
		return ;
	}
	// close file
	fileclose(fd) ;
	// create new file
	fd = filecreat("C:\\www\\cgi-bin\\output.txt") ;
	if(fd < 0)
	{
		// file creation error
		error("File Create Err") ;
		return ;
	}
	// add lines
	buf = buf + "One for the master,\r\n"
		+ "And one for the dame\r\n" ;
	// output to file
	rtn = filewrite(fd, buf, len(buf)) ;
	if(rtn < 0)
	{
		// file write error
		error("File Write Err") ;
	}
	// close file
	fileclose(fd) ;
}
```

8.4 Input/Output Binary File

The following sample script shows how to read a binary file and modify it, and overwrite the original file with the updated file.

```
Function main()
{
var     fd              ;       // file descriptor
var     data[ ]         ;       // data buffer
var     size            ;       // size of the file to write/load
var     rtn             ;

// open file
fd = fileopen("C:\\www\\cgi-bin\\test.bin", "rw") ;
if(fd < 0)
{
        error("File Open Err") ;
        return ;
}

// retrieve data into a hash
size = filereadhash(fd,&data[], getfilesize(fd)) ;
if(size < 0)
{
        error("File Read Err") ;
        fileclose(fd) ;
        return ;
}

// modify some part of data
data[2] = 42 ;          // 42 = 0x2A
data[4] = 0x5D ;

// move file access pointer to beginning of file
filelseek(fd, 0, 0) ;

// write into file
rtn = filewritehash(fd, data, size) ;
if(rtn < 0)
{
        error("File Write Err") ;
        fileclose(fd) ;
        return ;
}

// close file
fileclose(fd) ;
}
```

9.1 What is a Hash?

A hash is a kind of “container” in which you can manage more than one value (element) as a group.
In this respect, a hash is similar to an array. The difference is that the array specifies an element with an integer suffix, while the hash uses a string called a “key” to specify an element.

9.2 Hash Manipulation Script

The following script involves manipulation of a hash. It first defines a hash buffer and manipulates the buffer in various ways. A hash consists of indices, keys and values. An index is a value starting with zero, relevant to an array number of the array.

[Sample Script of Hash Manipulation]

```
//hash manipulation script
Function main()
{
var             rtn     ;
var             buf[ ]  ;
var             buf1[ ] ;
var             buf2[ ] ;

// define a hash
buf[“aaa”] = 10 ;
buf[“bbb”] = 20 ;
buf[“ccc”] = 30 ;
buf[“ddd”] = “abcd” ;
buf[“eee”] = “uxyz” ;


// get the count of hash items
rtn = gethashcount(buf) ;
puts(“Number of Hash Items: %rtn%\n”) ;//*1

// get the name specified for key
rtn = gethashkey(buf,1) ;
puts(“Specified Key Name:  %rtn%\n”) ;//*2

// check if key exists
rtn = ishashkey(buf, “bbb”) ;
if(rtn == -1)
{
        puts(“Specified key does not exist\n”) ;
}
else
{
        puts(“Specified key exists in index %rtn%\n”) ;//*3
}
```

index	key	value
0	aaa	10
1	bbb	20
2	ccc	30
3	ddd	″abcd″
4	eee	″uxyz″

```
// insert a hash item (in between “bbb” and “aaa”)
hashinsert(buf, “bbb”, 40) ;

// set key name for item
sethashkey(buf, 1, “ppp”) ;

// delete a hash item
hashdelete(buf, “ccc”) ;
puts(“Delete a Hash Item\n”) ;//*4
DspHash(buf) ;

// search for hash item with value of 20
rtn = hashfind(buf, 20) ;
puts(“Relevant Index: %rtn%\n”) ;//5
// search for hash item with value of “abcd”
rtn = hashfind(buf, “abcd”) ;
puts(“Relevant Index: %rtn%\n”) ;//6

// sort by hash key
hashkeysort(buf, 0, 1) ;
puts(“Sort by the Hash Key\n”) ;//*7
DspHash(buf) ;f

// sort by hash value in descending order
hashsort(buf, 1, 1) ;
puts(“Sort by the Hash Value\n”) ;//*8
DspHash(buf) ;

 hash items (copy three items starting with index 1)
 hashcopy(buf, 1, 3) ;
Copied Hash Items\n”) ;//*9
ash(buf1) ;

// merge (combine) hash tables
buf1[ ] = 0 ;    // initialize hash
buf1[“mmm”] = 60 ;
buf1[“nnn”] = 70 ;
buf2 = hashmerge(buf, buf1) ;
// “buf2[ ] = buf[ ] + buf1[ ] ;” returns same result
puts(“Merged Hash Table\n”) ;//*10
DspHash(buf2) ;
}

// a function to display hash
Function DspHash(hash)
{
var     i       ;
var     cnt     ;
var key         ;

// get the count of hash and loop the process for number of counts
cnt = gethashcount(hash) ;
for(i=0; i<cnt; i++)
```

index	key	value
0	aaa	10
1	ppp	40
2	bbb	20
3	ccc	30
4	ddd	″abcd″
5	eee	″uxyz″

index	key	value
0	aaa	10
1	ppp	40
2	bbb	20
3	ddd	″abcd″
4	eee	″uxyz″

index	key	value
0	aaa	10
1	bbb	20
2	ddd	″abcd″
3	eee	″uxyz″
4	ppp	40

index	key	value
0	eee	″uxyz″
1	ddd	″abcd″
2	ppp	40
3	bbb	20
4	aaa	10

index	key	value
0	eee	″uxyz″
1	ddd	″abcd″
2	ppp	40
3	bbb	20
4	aaa	10

index	key	value
0	eee	″uxyz″
1	ddd	″abcd″
2	ppp	40
3	bbb	20
4	aaa	10
3	mmm	60
4	nnn	70

```
{
        key = gethashkey(hash, i) ;
        puts(“[%i%](%key%)%hash[i]%\n”) ;
}
}
```

The result of the hash manipulation script should be as follows:

Note: Compare the content of the script to the displayed result based on the numbers shown beside.
The result should be the same as the figures.

[**Output Result**]

```
Number of Hash Items:  5                    *1
Specified Key Name:  bbb                    *2
Specified key exists in index 1             *3
Delete a Hash Item                          *4
[0](aaa)10
[1](ppp)40
[2](bbb)20
[3](ddd)abcd
[4](eee)uxyz
Relevant Index: 2                           *5
Relevant Index: 3                           *6
Sort by the Hash Key                        *7
[0](aaa)10
[1](bbb)20
[2](ddd)abcd
[3](eee)uxyz
[4](ppp)40
Sort by the Hash Value                      *8
[0](eee)uxyz
[1](ddd)abcd
[2](ppp)40
[3](bbb)20
[4](aaa)10
Copied Hash Items                           *9
[0](ddd)abcd
[1](ppp)40
[2](bbb)20
Merged Hash Table                           *10
[0](eee)uxyz
[1](ddd)abcd
[2](ppp)40
[3](bbb)20
[4](aaa)10
[5](mmm)60
[6](nnn)70
```

10. Using Regular Expressions

Regular expression is a method of expressing a pattern that can be used to search or replace various text strings. It is described with a combination of normal characters and symbols called “meta characters,” which have special meanings.
For example, the character (“^”) means “beginning of a line,” (“.”) represents “any one character,” and (“+”) represents “one or more occurrence of a character that precedes the expression.” Regular expression allows you to specify a “pattern” of the string you are looking for and because you can perform fuzzy search or batch replacement of different-looking strings, it is very useful when you cannot specify the exact string. In regular expression, the existence of a space or small differences like “computer” and “computers” is ignored.

10.1 Regular Expression Operators

The following table lists operators used in regular expressions:

No.	Regular Expression	Description
(1)	.	matches any one character
(2)	*	matches zero or more occurrences of one character
(3)	[]	matches any one of the characters within the brackets ([ ])
(4)	-	specifies range of continuous characters used in conjunction with the brackets of (3)
(5)	^	(a) negates the subsequent character or set of characters when used in a [ ] or at any position other than the beginning of a line. (b) anchors the match to the beginning of a line when used at the beginning of a line.
(6)	{m,n}	indicates m to n times occurrence of preceding character.
(7)	{n,m}	matches strings repeated *n* to *m* times.
(8)	$	anchors the match to the end of a line.
(9)	+	matches one or more occurrences of a character that precedes the expression (pattern.)
(10)	?	matches zero or one occurrence of a character that precedes the expression (pattern.)
(11)	|	indicates a choice between multiple items.
(12)	()	marks the beginning and the end of a subexpression.
(13)	\	negates the meaning of the subsequent meta character (regular expression operator.)
(14)	%~%	indicates a variable.
(15)	{n}	matches strings repeated *n* times.
(16)	{n,}	matches strings repeated at least *n* times.
(17)	*?	matches strings repeated zero or more times (non-greedy.)
(18)	+/	matches strings repeated one or more times (non-greedy.)
(19)	??	matches strings repeated zero time or once (non-greedy.)
(20)	{n,}?	matches strings repeated at least *n* times (non-greedy.)
(21)	{n,m}?	matches strings repeated *n* to *m* times (non-greedy.)

Notes on “non-greedy”:

For example, a search process for the string “abcdAB12abcd” with regular expression of “abc.*d” (greedy) begins by seeking the first four characters “abcd”, and then continues to search for the subsequent characters. It continues to search until the last character “d” matches. The result will be “abcdAB12abcd.” On the other hand, regular expression of “abc.*?d” (non-greedy) ends searching when the first four characters “abcd” matches. Consequently, the result will be “abcd.”

10.2 Basic Examples

\	Search for “abcd” regmatch(“abcd”, “abcdABCD12abcd”, hash[ ]) ; regmatchall(“abcd”, “abcdABCD12abcd”, hash[ ]) ;	 hash[0]=”abcd” hash[0]=”abcd” hash[1]=”abcd”
^	Search for “abcd” at the beginning of the string regmatch(“^abcd”, “abcdABCD12abcd”, hash[ ]) ; regmatchall(“^abcd”, “abcdABCD12abcd”, hash[ ]) ;	 hash[0]=”abcd” Will get the same result
$	Search for “abcd” at the end of the string regmatch(“abcd$”, “abcdABCD12abcd”, hash[ ]) ; regmatchall(“abcd$”, “abcdABCD12abcd”, hash[ ]) ;	 hash[0]=”abcd” Will get the same result
.	Search for any one character regmatch(“.”, “abcd”, hash[ ]) ; regmatchall(“.”, “abcd”, hash[ ]) ;	 hash[0]=”a” hash[0]=”a” hash[1]=”b” hash[2]=”c” hash[3]=”d”
| ()	Case 1: Search for “abcd” without regard to case regmatch(“(A|a)(B|b)(C|c)(D|d)”, “AbcD”, hash[ ]) ; regmatchall(“(A|a)(B|b)(C|c)(D|d)”, “AbcD”, hash[ ]) ;	hash[0]=”AbcD” Will get the same result
[ ] {n}	Case 2: Search for “abcd” without regard to case regmatch(“[a-zA-Z]{4}”, “AbcD”, hash[ ]) ; regmatchall(“[a-zA-Z]{4}”, “AbcD”, hash[ ]) ;	 hash[0]=”AbcD” Will get the same result
*	Search for zero or more occurrence of any one character “.” regmatch(“abc.*d”, “abcd”, hash[ ]) ; regmatchall(“abc.*d”, “abcd”, hash[ ]) ;	 hash[0]=”abcd” Will get the same result
+	Search for one or more occurrences of any one character “.” regmatch(“abc.+d”, “abcd”, hash[ ]) ; regmatchall(“abc.+d”, “abcd”, hash[ ]) ;	 No match found (to get a match, you need strings like “abc1d” or “abcABCd”) Will get the same result
?	Search for zero or one occurrences of any one character “.” regmatch(“abc.?d”, “abcd”, hash[ ]) ; regmatch(“abc.?d”, “abc1d”, hash[ ]) ; regmatch(“abc.?d”, “abc12d”, hash[ ]) ; regmatchall(“abc.?d”, “abcd”, hash[ ]) ; regmatchall(“abc.?d”, “abc1d”, hash[ ]) ; regmatchall(“abc.?d”, “abc12d”, hash[ ]) ;	 hash[0]=”abcd” hash[0]=”abc1d” No match found Will get the same result
{n} {n,} {n,m}	Difference in the number of occurrences (specifying alphabets) regmatch(“[a-zA-Z]{4}”, “abcdAB”, hash[ ]) ; regmatch(“[a-zA-Z]{4,}”, “abcdAB”, hash[ ]) ; regmatch(“[a-zA-Z]{4,5}”, “abcdAB”, hash[ ]) ; regmatchall(“[a-zA-Z]{4}”, “abcdAB”, hash[ ]) ; regmatchall(“[a-zA-Z]{4,}”, “abcdAB”, hash[ ]) ; regmatchall(“[a-zA-Z]{4,5}”, “abcdAB”, hash[ ]) ;	 hash[0]=”abcd” hash[0]=”abcdAB” hash[0]=”abcdA” Will get the same result

*?	Difference between “*” and “*?” regmatch(“abc.*d”, “abcdAB12abcd”, hash[ ]) ; regmatch(“abc.*?d”, “abcdAB12abcd”, hash[ ]) ; (the latter stops searching upon meeting the first “d”) regmatchall(“abc.*d”, “abcdAB12abcd”, hash[ ]) ; regmatchall(“abc.*?d”, “abcdAB12abcd”, hash[ ]) ;	 hash[0]=” abcdAB12abcd” hash[0]=” abcd” hash[0]=” abcdAB12abcd” hash[0]=” abcd” hash[1]=” abcd”-
+?	Difference between “+” and “+?” regmatch(“ab.+d”, “abcdAB12abcd”, hash[ ]) ; regmatch(“ab.+?d”, “abcdAB12abcd”, hash[ ]) ; (the latter stops searching upon meeting the first “d”) regmatchall(“ab.+d”, “abcdAB12abcd”, hash[ ]) ; regmatchall(“ab.+?d”, “abcdAB12abcd”, hash[ ])	 hash[0]=” abcdAB12abcd” hash[0]=” abcd” hash[0]=” abcdAB12abcd” hash[0]=” abcd” hash[1]=” abcd”
??	Difference between “?” and “??” regmatch(“abc.?d”, “abcdd”, hash[ ]) ; regmatch(“abc.??d”, “abcdd”, hash[ ]) ; (the latter stops searching upon meeting the first “d”) regmatchall(“abc.+d”, “abcdAB12abcd”, hash[ ]) ; regmatchall(“abc.+?d”, “abcdAB12abcd”, hash[ ]) ;	 hash[0]=” abcdd” hash[0]=” abcd” Will get the same result
{n,}?	Difference between “{n,}” and “{n,}?” regmatch(“[a-zA-Z]{4,}”, “abcdAB”, hash[ ]) ; regmatch(“[a-zA-Z]{4,}?”, “abcdAB”, hash[ ]) ; regmatchall(“[a-zA-Z]{4,}”, “abcdAB”, hash[ ]) ; regmatchall(“[a-zA-Z]{4,}?”, “abcdAB”, hash[ ]) ;	 hash[0]=”abcdAB” hash[0]=”abcd” Will get the same result

10.3 Advanced Examples

1. If the string includes characters other than SBCS alphanumerical, it will return 1, as in ([^] is an indication of “not”):

```
rtn = regmatch("[^a-zA-Z0-9]","abcd=efgh", list[ ]) ;
```

2. If the string (mailAddr) is an E-mail address, it will return 1, as in:

```
rtn = regmatch("^[^@]+@[^.]+\..+", mailAddr, list[ ]) ;
```

3. If the string (url) is a URL starting with “http://” or “ftp://”, it will return 1, as in:

```
rtn = regmatch("^(http|ftp)://[^.]+\..+", url, list[ ]) ;
```

11. Invoking Events

Generally, when utilizing ProDigio as a Web server, you will not need to manually invoke script from standard input (debug monitor.) It is only for maintenance that you will need to execute maintenance commands or scripts from standard input (debug monitor.) While ProDigio is under actual operation, scripts run only when they are invoked by CGI, SSI, or events.

11.1 Registering Events

Use the **event** function to register events. With this function, you specify the event type, the name of scripts to execute, the requirements of invoking events, and so on. For more information on functions, see Chapter 13

11.2 Stopping Registered Events

You can also use the **event** function to stop events that you have registered. Note that an event that matches a boot function name will also be stopped.
[On Registration] Specify to start test.wcr at zero second every minute.
event(“RTC”,”C:\\www\\test.wcr”,””,””,”0”,-1, 1000) ;
[On Stopping] Change the 6th argument to zero.
event(“RTC”,”C:\\www\\test.wcr”,””,””,”0”,0, 1000) ;

11.3 Autorun Script

This is a script that runs only once, when ProDigio is started. The script is a file located in C:\system\autoscript.wcr. You must register all events required for ProDigio operation with this script before you actually start using ProDigio. Otherwise, the maintenance personnel will end up having to invoke the script every time ProDigio restarts.

11.4 Parameter String (Argument)

The “parameter string” is the third argument of **event** function. The parameter string can pass more than one parameter on to the startup script by separating them with a space character.

[Example]
event(“RTC”,”C:\\www\\test.wcr”,”1 a=2 abc”,””,”0”,-1, 1000) ;

In this case, test.wcr will have the following values: arg[0]=”1”, arg[1]=”a=2”, and arg[2]=”abc.”

11.5 Event Types

11.5.1 TIMER Event

This event runs a script at a fixed interval or after a specified period of time has passed. When invoking a script after a specified period of time, the time count starts from the point the function is called.

11.5.2 RTC Event

This event can invoke scripts at any time.

[Example]

1. Specify the time and exact date including the year (launch at 12:34:56 on January 26, 2004)
   event(“RTC”, “C:\\www\\aaa.wcr”, “”, “12:34:56”, “04/1/26”, ...
2. Specify the time and date (launch at 12:34:56 on January 26 every year)
   event(“RTC”, “C:\\www\\aaa.wcr”, “”, “12:34:56”, “1/26”, ...
3. Specify the time and the date of the month (launch at 12:34:56 on 26th every month)
   event(“RTC”, “C:\\www\\aaa.wcr”, “”, “12:34:56”, “26”, ...
4. Specify the time (launch at 12:34:56 every day)
   event(“RTC”, “C:\\www\\aaa.wcr”, “”, “12:34:56”, “”, ...
5. Specify the second and minute (launch at 34:56 every hour)
   event(“RTC”, “C:\\www\\aaa.wcr”, “”, “34:56”, “”, ...
6. Specify the second (launch at 56 second every minute, e.g. 0:56, 1:56, 2:56 ...)
   event(“RTC”, “C:\\www\\aaa.wcr”, “”, “56”, “”, ...

11.5.3 SIO Event

This event invokes a script when it receives the specified number of bytes from the serial ports (COM1/COM2.)

◆ **Note**: Do not register more than one SIO event for fear that unexpected SIO events may occur.

11.5.4 PIO Event

This event invokes a script when any signal at a PIO terminal is changed. Make sure to use **setport** function and set the target PIO terminal mode to input mode

11.5.5 PULSE Event

This event retrieves the number of times that a PIO changed within a specified period of time and invokes scripts.
The count of PIO changes (the number of PULSE) is obtained via the **getpiopulse** function in the launched script.
If necessary, clear the number of PULSE using the **clrpiopulse** function.

```
[Example]
Script A
// PULSE event (startup script: test.wcr, no argument specified, PIO = port 0, monitor every
second)
event("PULSE","C:\www\test.wcr","",0,1000,-1,5000,1) ;

Script B (test.wcr)
rtn = getpiopulse(0) ;  // obtain number of pulses on port 0.
clrpiopulse(0) ;        // clear number of pulses.
```

11.5.6 USER Event

This event fires the specified script after the currently executed event is terminated.

11.6 What is an Event Queue?

If multiple events occurred at the same time or an event occurred while a script was being executed, the events will be stored in an event management table inside ProDigio, called an event queue. The status of events changes to "wait" upon entering the event queue and the event with the highest priority in the queue will be executed first after the currently running script finishes. (If any events with the same priority exist in the queue, they will be run in the order they were registered.)

11.7 Priority of Events

If events occur simultaneously, the **event** function normally runs in the order of their registration. However, it is possible to set a particular event so that it has the highest priority or a higher priority than another script

```
event("TIMER", "C:\\www\\test1.wcr", "",1000,0,-1,1000) ;
                    // 5 is assumed when the priority is not specified
event("TIMER", "C:\\www\\test1.wcr", "",5000,0,-1,1000,1) ;
                    // register an event with the highest priority by setting the priority value to 1
```

For example, if you want to run a USER script in preference to any other **events** for some reason, you can override the preceding TIMER event by giving the USER script the maximum priority value of one and register its event function as shown in the second example above.
However, this is available only when events occurred at the same time or more than one event is queued in the event queue waiting to be executed

11.8 Notes

If the startup script registers the same events more than once, the event registered at a later time holds good.

[Example] If you write scripts as follows, the Event 2 will be validated:

```
event(“TIMER”, “C:\\www\\test1.wcr”, “”, 1000, 0, -1, 1000) ;// Event 1
event(“TIMER”, “C:\\www\\test1.wcr”, “”, 5000, 0, -1, 1000) ;// Event 2
```

Also, if the startup script in different scripts register the same event, the last registered event holds good.

Script A

```
event(“TIMER”, “C:\\www\\test1.wcr”, “”, 1000, 0, -1, 1000) ;// Event 1
```

Script B

```
event(“TIMER”, “C:\\www\\test1.wcr”, “”, 5000, 0, -1, 1000) ;// Event 2
```

◆ **Notes on Event Timeout:**

The seventh argument of the **event** function is “monitoring time.” If a script is trapped in an infinite loop, other scripts cannot be invoked and the ProDigio-installed device stops functioning. To avoid such a situation, you need to set limits on script execution. It is “monitoring time.” ProDigio is designed to be restarted when the monitoring time has been exceeded. For this reason, please keep in mind that only one script or CGI/SSI can run at one time and set appropriate values for script execution time and monitoring time.

12. Sending E-mails

With ProDigio WSC script, you can send E-mails with ProDigio's SMTP E-mail server feature.
In addition, it is possible to send not only the body of a message but also attachments*1.
This feature can be used for various purposes such as sending warning notification messages, or sending E-mails with data obtained from external devices.

This section shows a sample script for sending an E-mail.

Send To: xxxx@xxxxxx.com
Subject: Delivery test
Body: This is an e-mail delivery test with SMTP feature.
Sender: yyyy@yyyyyy.com
Attachment name: test.txt

The script to send an E-mail with the above contents will be as follows:

```
rtn = sendmail("yyyy@yyyyyy.com","xxxx@xxxxxx.com","test",
    "sendmailtest","test.txt");
```

*1 The maximum size of attachments that can be sent with an E-mail is 1 MB.

To actually send the E-mail, you need to modify the script in the following way:

```
Function main()
{
rtn = sendmail("yyyy@yyyyyy.com","xxxx@xxxxxx.com","test",
    "sendmailtest","test.txt");
}
```

To provide you with a more realistic example, you can write an **if-else** statement that executes an **else** statement when the specified range of number is exceeded and include this **sendmail** function in the **else** statement so that a warning message will be sent to the appropriate user to let him/her know that the limit has been exceeded. You can use this script in many ways, such as for voltage, temperature.

12.1 Sending E-mails with sendmail Function from Browser

This section provides a sample script for sending an E-mail from a browser based on the information entered by using the E-mail delivery feature of ProDigio.
The following is a sample script from “6. Working with CGI and SSI”; it runs WSC script with CGI feature and retrieves arguments from the browser to send an E-mail.

[Script File]

[sendmail.wcr]

```
Function main()
{
var afrom;
var ato;
var asub;
var ames;
var afiles;

afrom  = right(arg[0],len(arg[0])-len(“from=”));   // send from
ato    = right(arg[1],len(arg[1])-len(“to=”));     // send to
asub   = right(arg[2],len(arg[2])-len(“sub=”));    // subject
ames   = right(arg[3],len(arg[3])-len(“mes=”));    // body
afiles = right(arg[4],len(arg[4])-len(“files=”));  // attachment

rtn    =sendmail(“%afrom%”,”%ato%”,”%asub%”,“%ames%”,”%afiles%”);
sleep(50);

webprint (“<html>\r\n”);
webprint (“<html><head><title>E-mail Sent Successfully </title>\r\n”);
webprint (“<META HTTP-EQUIV=\”Refresh\” CONTENT=\”5\”;
          URL=sendmail.html”></head>\n”);
webprint (“<body>\r\n”);
webprint (“<br>\r\n”);
webprint (“Send from: %afrom%<br>\r\n”);
webprint (“Send to: %ato%<br>\r\n”);
webprint (“Subject: %asub%<br>\r\n”);
webprint (“Body: %ames%<br>\r\n”);
webprint (“Attachment: %afiles%<br>\r\n”);
webprint (“<br><br>Send to: %ato%\r\n”);
webprint (“An e-mail with the preceding contents has been sent to the above
          recipient.\r\n”);
webprint (“<br>\r\n”);
webprint (“<br>\r\n”);
webprint (“Will be redirected to the previous page in 5 seconds.\n”);
webprint (“</body>\r\n”);
webprint (“</html>\r\n”);

}
```

[HTML File]

[sendmail.html]

```
<html>
<head>
<meta http-equiv="Content-Language" content="ja">
<meta http-equiv="Content-Type" content="text/html; charset=iso8801">
<title>E-mail Submission Form</title>
</head>
<body>
<p>E-mail Submission Form</p>
<form method="POST" action="sendmail.wcr">
<p>Source address: <input type="text" name="from" size="40"></p>
<p>Destination address: <input type="text" name="to" size="40"></p>
<p>Subject</p>
<input type="text" name="sub" size="40"></p>
<p>Body<br>
<textarea rows="7" name="mes" cols="67"></textarea></p>
<p>Attachment<br>
<textarea rows="7" name="file" cols="67"></textarea></p>
<p><input type="submit" value="Submit">
<input type="reset" value="Reset"></p>
</form>
<br>
</body>
</html>
```

13. About Functions

This chapter provides an overview and examples of functions provided in WSC script.
You can combine these functions to create a script that meets your needs

Section No.	Function Symbol Name	Description
1.Common Functions		
13-1-1	error()	Outputs strings (error message) to standard output
13-1-2	istype()	Gets the type (INTEGER, STRING, or FLOAT)
13-1-3	int()	Converts a numerical string to a numerical value (hexadecimal, octal, decimal , or floating point)
13-1-4	uintstr()	Converts a numerical value to an unsigned numerical string
13-1-5	sleep()	Waits for the specified period of time (in msec)
13-1-6	hexstr()	Converts a numerical value to an unsigned hexadecimal numerical string
2.Time Functions		
13-2-1	getrtctime()	Gets the current time
13-2-2	setrtctime()	Sets time
13-2-3	getdatestring()	Gets date string
13-2-4	gettimestring()	Gets time string
13-2-5	difftime()	Calculates the difference between the specified time
13-2-6	getsntptime()	Gets time from SNTP
13-2-7	statsntp()	Views the status of SNTP
3.String Manipulation Functions		
13-3-1	chr()	Converts a numerical value to a character code
13-3-2	asc()	Converts a character code to a numerical value
13-3-3	string()	Lists the specified number of specified characters
13-3-4	trim()	Removes spaces in front or subsequent of a string
13-3-5	len()	Gets the length of a string
13-3-6	strcmp()	Compares strings (including the specification of comparison range)
13-3-7	strchr()	Detects the first appearance position of the specified character
13-3-8	strrchr()	Detects the last appearance position to the specified character
13-3-9	strstr()	Detects the first appearance position of the specified string
13-3-10	strmatch()	Compares strings using wildcards
13-3-11	left()	Extracts the specified number of characters from left side of the string
13-3-12	right()	Extracts the specified number of characters from right side of the string
13-3-13	mid()	Extracts the specified number of characters from the specified position of the string
13-3-14	instr()	Detects the position of the specified string from the point of specification
13-3-15	tolower()	Converts alphabet string to lower case
13-3-16	toupper()	Converts alphabet string to upper case
4.Hash Manipulation Functions		
13-4-1	gethashcount()	Gets the count of hash items
13-4-2	sethashcount()	Changes the number of hash elements
13-4-3	gethashkey()	Gets the key name at the specified position in a hash
13-4-4	sethashkey()	Sets the key name to the specified position in a hash
13-4-5	ishashkey()	Checks the existence of a key in the hash
13-4-6	hashinsert()	Inserts a hash item
13-4-7	hashdelete()	Deletes a hash item
13-4-8	hashfind()	Finds hash contents
13-4-9	hashkeysort()	Sorts a hash by the key
13-4-10	hashsort()	Sorts a hash by the contents
13-4-11	hashcopy()	Copies the contents of a hash
13-4-12	hashmerge()	Merges multiple hash tables
13-4-13	hashtostring()	Converts a hash to a string
13-4-14	stringtohash()	Expands a string to a hash
5.Regular Expressions-Enabled Functions		
13-5-1	regmatch()	
13-5-2	regmatchall()	
13-5-3	regreplace()	

6. File Manipulation Functions		
13-6-1	getcurdir()	Gets the current directory
13-6-2	setcurdir()	Sets the current directory
13-6-3	dirmake()	Creates a directory
13-6-4	dirremove()	Deletes a directory
13-6-5	diropen()	Opens a directory
13-6-6	dirclose()	Closes a directory
13-6-7	dirread()	Gets a file name from a directory
13-6-8	filecreat()	Creates a file
13-6-9	fileopen()	Opens a file
13-6-10	fileclose()	Closes a file
13-6-11	fileread()	Reads strings from a file
13-6-12	filewrite()	Writes strings into a file
13-6-13	filereadhash()	Loads strings from a file into an associative array
13-6-14	filewritehash()	Writes data from an associative array into a file
13-6-15	getfilesize()	Gets the file size
13-6-16	filedelete()	Deletes a file
13-6-16	filerename()	Renames a file
13-6-18	filecopy()	Copies a file
13-6-19	filetell()	Gets the file access position
13-6-20	fileseek()	Moves the file access position
13-6-21	filemod()	Changes the file attribute
13-6-22	fputs()	Outputs one line string
13-6-23	fgets()	Gets one line string
13-6-24	fflush()	Refresh the flash file (C:\) drive.
7. Standard I/O Functions		
13-7-1	puts()	Outputs strings to standard output
13-7-2	gets()	Gets strings from standard input
13-7-3	getchar()	Gets a character from standard input
13-7-4	beep_on	Turns beep on
13-7-5	beep_off	Turns beep off
8. SIO I/O Functions		
13-8-1	sioputs()	Outputs strings to the serial port
13-8-2	siogets()	Gets strings from the serial port
13-8-3	sioputb()	Outputs binary data to the serial port
13-8-4	siogetb()	Gets binary data from the serial port
13-8-5	sioctrl()	Sets the serial port communication speed
13-8-6	siorxclr()	Clears the serial receive buffer
13-8-7	sioputd()	Outputs a binary data array to the serial port
13-8-8	siogetd()	Gets a binary data array from the serial port
13-8-9	siotxclr()	Clears the serial transmit buffer
13-8-10	siosigctrl()	Control the serial number string (RTS,DTR)
13-8-11	getsiosig()	Get the serial number string(CD,DSR,CTS)value.
13-8-12	com1_dtr_out()	Control the serial number string DTR from COM1
13-8-13	com1_ptn_in()	Get the serial number string(DTR,DSR,DCD)value from COM1.
13-8-14	com1_ctrl()	Control the serial number string RTS/CTS from COM1.
9. PIO I/O Functions		
13-9-1	putport()	Outputs 1 bit data to PIO port
13-9-2	getport()	Gets 1 bit data to PIO port
13-9-3	putpio()	Outputs1 bit data to multiple PIO ports
13-9-4	getpio()	Gets 1 bit data to multiple PIO ports
13-9-5	setport()	Sets the I/O direction of a PIO port
13-9-6	setpio()	Sets the I/O direction of multiple PIO ports
13-9-7	getpiopulse()	Gets the count of PIO pulse
13-9-8	clrpiopulse()	Clears the count of PIO pulse
13-9-9	statport()	Gets the I/O direction of a PIO port
13-9-10	statpio()	Gets the I/O direction of multiple PIO ports
13-9-11	putexport()	Outputs 1 bit data to an extended multipurpose PIO port
13-9-12	getexport()	Gets 1 bit data from an extended multipurpose PIO port
13-9-13	putexpio()	Outputs 1 bit data to extended multipurpose PIO ports

13-9-14	getexpio()	Gets 1 bit data from extended multipurpose PIO ports
13-9-15	setexport()	Set the I/O direction of an extended multipurpose PIO port
13-9-16	setexpio()	Sets the I/O direction of extended multipurpose PIO ports
13-9-17	statexport()	Gets the I/O direction of an extended multipurpose PIO port
13-9-18	statexpio()	Gets the I/O direction of extended multipurpose PIO ports
10. AD Conversion Function		
13-10-1	getadc()	Gets an A/D conversion value
11. I2C I/O Functions		
13-11-1	i2cput()	Writes data into IIC
13-11-2	i2cget()	Reads data from IIC
13-11-3	i2ci2cset()	Sets IIC device information
13-11-4	i2cwrite()	Outputs multi-byte of contiguous data to IIC
13-11-5	i2cread()	Gets multi-byte of contiguous data from IIC
12. FTP I/O Functions		
13-12-1	ftpput()	Sends a file via FTP
13-12-2	ftpget()	Receives a file via FTP
13-12-3	stat_ftp()	Gets the status of FTP Send/Receive
13. Send E-Mail Functions		
13-13-1	sendmail()	Sends an E-mail
13-13-2	stat_smtp()	Gets the status of E-mail Send/Receive
14. WEB Operation Function		
13-14-1	webprint()	Outputs data to the WEB browser
15. Event Process Function		
12-15-1	event()	Registers events
16 System Control Functions		
13-16-1	reboot()	Reboots the system
13-16-2	getwscparam()	Reads a parameter
13-16-3	setwscparam()	Sets a parameter
13-16-4	storewscparam()	Stores a parameter in flash ROM
13-16-5	crtwscparamfile()	Creates a parameter file
13-16-6	reiniwscparam()	Resets to the factory default value
13-16-7	getmacadr()	Gets the MAC address
13-16-10	phpowctrl	Controls the power supply
13-16-11	get_module_sw	Gets DIP-SW value on the module
13-16-12	outevtlogmsg	Outputs a message in the event log
13-16-13	getsysver()	Get the system version information character string
17. Dialing Control Function		
13-17-1	start_atdialup	Opens an ISP/SMTP dialup connection
13-17-2	stop_atdialup	Disconnects the communication line
13-17-3	stat_atconnect	Gets the status of the line
13-17-4	setini_atcommand	Sets the initialization AT command
13-17-5	setdial_atcommand	Sets the dialup AT command
13-17-6	get_atispno	Gets the line type
13-17-7	get_atresult	Gets the error result code on connection
13-17-8	seterr_atresult	Sets the error result code on connection

Layout of Function Specifications

Section No. Function Name Brief Description

[Description]
This section provides a usage overview of functions

[Syntax]
The syntax is the pattern you need to follow when writing scripts.
When writing your own script, comply with the instructions provided in this section

[Parameters]
A parameter is an “element” or “condition” accompanying various things.
For example, when driving a car, “speed,” “engine revolutions,” and “diameter of tire” etc., can be referred to as “parameters.”
In case of scripts, parameters are called “arguments.”

[Return Value]
A return value is a result value that the function returns to the caller.

[Example]
This section contains sample scripts.

[Remarks]
This section contains other reference information and notes.

13.1 Common Functions

13.1.1 error() Output Strings (Error Message) to Standard Output

[Description]
Outputs strings (error message) to standard output and the event log

[Syntax]
error(buf) ;

[Parameters]
buf Error message strings (only alpha/numerical SBCS are allowed)

[Return Value]
Returns a value of 0.

[Example]

```
error("### ERROR ###\n") ;
```

[Remarks]
error() function is different from **puts()** function in that its output strings are written into the event log.

13.1.2 **istype()** Get Type (INTEGER, STRING, or FLOAT)

[Description]

Gets the type (INTEGER, STRING, or FLOAT) of a specified variable.

[Syntax]

```
type = istype(val) ;
```

[Parameters]

val	A variable whose type is to be checked.

[Return Value]

type	Success: returns one of the following numbers according to the identified type: 0: Integer (INTEGER) 1: String (STRING) 2: Floating point (FLOAT)

[Example]

1. INTEGER

```
type = istype(100) ;                    // type=0
```

2. STRING

```
type = istype(“abc”) ;                  // type=1
```

3. FLOAT

```
type = istype(5.63) ;                   // type=2
```

13.1.3 int() Convert Numerical String to Numerical Value (Octal, Decimal, Hexadecimal, or Floating Point)

[Description]

Converts a numerical string to a numerical value (octal, decimal, hexadecimal, or floating point.)

[Syntax]

```
val = int(str) ;
```

[Parameters]

str	A numerical string whose type is to be converted to INTEGER (use one of the following formats as appropriate) Octal notation: “0nnn” — A numerical string starting with a value of “0” Decimal notation: “nnnn” — A numerical string starting with any number other than “0” Hexadecimal notation: “0xmmm” or “0Xmmm” — Hexadecimal (0 - 9, A - F, a - f) string starting with “0x” or “0X” Floating point notation: “nn.nn” — A numerical string containing ‘.’

[Return Value]

val	Success: returns the numerical value converted from a numerical string **Note**: If the string contains any invalid characters, this function converts the string up to the point where an invalid character exists.

[Example]

1. Octal String

```
val = int(“077”) ;          // val = 63
```

2. Decimal String

```
val = int(“123”) ;          // val = 123
```

2. Hexadecimal String

```
val = int(“0x4a”) ;         // val = 74
```

4. Floating Point String

```
val = int(“32.56”) ;        // val = 32.560000
```

13.1.4 **uintstr()** Convert Numerical Value to Unsigned Digit String

[Description]

Converts an unsigned numerical value (octal, decimal, hexadecimal, or floating point) to a string.

[Syntax]

str = uintstr(i) ;

[Parameters]

i	A numerical value to be converted (use one of the following formats as appropriate) Octal numerical value: 0nnn — A numerical string starting with a value of '0' Decimal numerical value: nnnn — A numerical string starting with any num ber other than '0' Hexadecimal numerical value: 0xmmm or 0Xmmm — Starting with 0x or 0X Floating point numerical value: nn.nn — A numerical string containing '.'

[Return Value]

str	Success: returns the string converted from an unsigned numerical value **Note**: If a floating point is specified, the fractional part will be truncated.

[Example]

1. Octal Number

```
str = uintstr(077) ;          // str="63"
```

2. Decimal Number

```
str = uintstr(160) ;          // str="160"
```

3. Hexadecimal Number

```
str = uintstr(0x80) ;         // str="128"
```

4. Floating Point

```
str = uintstr(10.56) ;        // str="10"
```

13.1.5 **sleep()** Wait for Specified Period of Time (in msec)

[Description]
Makes the process wait for a specified period of time (in seconds.)

[Syntax]
sleep(tms) ;

[Parameters]
tms Specifies the wait time in milliseconds.

[Return Value]
No return value

[Example]

```
sleep(100) ;                // wait for 100ms
```

[Remarks]
The arguments section says to set the value in milliseconds, however, the resolution is 10 milliseconds and any smaller values will be truncated. If you specify “sleep(105);”, 5 will be ignored and the value passed onto the function will be “100.”
Note that there could be some delay in the specified time up to 10 milliseconds.

13.1.6 **hexstr()** Convert Numerical Value to Unsigned Hexadecimal Numerical String

[Description]

Converts an unsigned numerical value (octal, decimal, hexadecimal, or floating point) to a hexadecimal numerical string.

[Syntax]

```
str = hexstr(x) ;
```

[Parameters]

x	A numerical value to be converted (use one of the following formats as appropriate) Octal numerical value: 0nnn — A numerical string starting with a value of '0' Decimal numerical value: nnnn — A numerical string starting with any number other than '0' Hexadecimal numerical value: 0xmmm or 0Xmmm — A numerical string starting with 0x or 0X Floating point numerical value: nn.nn — A numerical string containing '.'

[Return Value]

str	Success: returns the converted unsigned hexadecimal string **Note**: If the floating point is specified, the fractional part will be truncated.

[Example]

1. Octal Number

```
str = hexstr(077) ;          // str="3f"
```

2. Decimal Number

```
str = hexstr(160) ;          // str="a0"
```

3. Hexadecimal Number

```
str = hexstr(0x80) ;         // str="80"
```

4. Floating Point

```
str = hexstr(10.56) ;        // str="a"
```

13.2 Time Functions

13.2.1 **getrtctime()** Get Current Time

[Description]
Gets the current time (date and time) from RTC (built-in clock) in ProDigio.

[Syntax]
t = getrtctime();

[Parameters]
No parameters

[Return Value]
t Success: returns an associative array representing the current time
t[“year”] - Year (2xxx)
t[“month”] - Month (1 - 12)
t[“day”] - Day (1 - 31)
t[“hour”] - Hour (0 - 23)
t[“minute”] - Minute (0 - 59)
t[“second”] - Second (0 - 59)

Failure: returns the following values when the function failed to retrieve the current time:
t[“year”] :0
t[“month”] :0
t[“day”] :0
t[“hour”] :0
t[“minute”] :0
t[“second”] :0

[Example]

```
var t ;
t = getrtctime() ;   //  t[“year”] = 2004
                     //  t[“month”] = 1
                     //  t[“day”] = 20
                     //  t[“hour”] = 9
                     //  t[“minute”] = 10
                     //  t[“second”] = 15
```

13.2.2 **setrtctime()** Set Time

[Description]
Sets the time (date and time) to RTC (built-in clock) in ProDigio.

[Syntax]
rtn = setrtctime(t) ;

[Parameters]

t	An associative array representing the time

[Return Value]

rtn	Success: returns 0 Failure: returns -1 (setting time is not allowed)

[Example]

```
var t ;
t[“year”] = 2004 ;
t[“month”] = 1 ;
t[“day”] = 20 ;
t[“hour”] = 9 ;
t[“minute”] = 10 ;
t[“second”] = 15 ;
rtn = setrtctime(t) ;                    // set the time to 9:10:15 on January 20, 2004
```

13.2.3 **getdatestring()** Get Date String

[Description]
Gets date string in the specified format.

[Syntax]
datestr = getdatestring(format,t) ;

[Parameters]

format	The format to use for date string conversion (“yyyy/MM/dd”) Delimiters: “/” or space y Two-digit year without leading 0 (2004-->4, 2014-->14) yy Two-digit year yyyy Four-digit year M Month without leading 0 MM Month with leading 0 MMM Month represented in abbreviated form MMMM Month represented by string d Day without leading 0 dd Day with leading 0
t	An associative array representing time. The current date is assumed when this parameter is not specified.

[Return Value]

datestr	Success: returns a date string in the specified format Failure: returns a NULL string (“ “)

[Example]

1. Time Specified

```
t[“year”] = 2004 ;
t[“month”] = 1 ;
t[“day”] = 20 ;
t[“hour”] = 9 ;
t[“minute”] = 10 ;
t[“second”] = 15 ;
datestr =  getdatestring(“yy/MM/dd”,t) ;        // datestr = “04/01/20”
```

2. Time Not Specified

```
datestr =  getdatestring(“yy/MM/dd”) ;          // datestr: current_date
```

13.2.4 **gettimestring()** Get Time String

[Description]
Gets a time string in the specified format.

[Syntax]
timestr = gettimestring(format,t)

[Parameters]

format	The format to use for time string conversion (“*HH:mm:ss*”)	
	Delimiter: ‘ : ‘, or space	
	h	Hour in 12-hours format without leading 0
	hh	Hour in 12-hours format with leading 0
	H	Hour in 24-hours format without leading 0
	HH	Hour in 24-hours format with leading 0
	m	Minute without leading 0
	mm	Minute with leading 0
	s	Second without leading 0
	ss	Second with leading 0
t	An associative array representing time. The current time is assumed if this parameter is not specified.	

[Return Value]

timestr	Success: returns a time string in the specified format
	Failure: returns a NULL string (“ “)

[Example]

1. Time Specified

```
t[“year”] = 2004 ;
t[“month”] = 1 ;
t[“day”] = 20 ;
t[“hour”] = 9 ;
t[“minute”] = 10 ;
t[“second”] = 15 ;
timestr = gettimestring(“HH:mm:ss”,t) ;          // timestr = “09:10:15”
```

2. Time Not Specified

```
timestr = gettimestring(“HH:mm:ss”) ;          // timestr: current_time
```

13.2.5 **difftime()** Get Time Difference

[Description]

Returns the differential between a specified pair of times (including second, minute, hour, day, month, and year) in seconds.
The supported time is from year 2000 onwards.

[Syntax]

```
diffsec = difftime(t1,t2) ;
```

[Parameters]

t1	An associative array representing time
t2	An associative array representing time

[Return Value]

diffsec	Success: returns the differential of |t1-t2| in seconds as an absolute value (0 or more) Failure: returns -1 (invalid date, month, and/or year)

[Example]

```
t1[“year”] = 2001 ;
t1[“month”] = 1 ;
t1[“day”] = 15 ;
t1[“hour”] = 18 ;
t1[“minute”] = 0 ;
t1[“second”] = 0 ;

t2[“year”] = 2000 ;
t2[“month”] = 12 ;
t2[“day”] = 31 ;
t2[“hour”] = 18 ;
t2[“minute”] = 0 ;
t2[“second”] = 0 ;
diffsec = difftime(t1,t2) ;          // diffsec = 1296000
```

13.2.6 **getsntptime()** Get Time from SNTP

[Description]
Gets time from SNTP and sets it to RTC (built-in clock) in ProDigio.
(Gets time regardless of SNTP status flag set in the system.)

[Syntax]
rtn = getsntptime(wt) ;

[Parameters]

wt	Connection wait time in msec **Note**: Specify on a 1,000 msec basis because its value is rounded down to the thousands.

[Return Value]

rtn	Success: returns a value of 0 or more Failure: returns a value less than 0 **Note**: For more information on error codes, see page 234.

[Example]

```
rtn = getsntptime(5000) ;          // connection wait time is 5 seconds
```

13.2.7 **statsntp()** View SNTP Status

[Description]
Views the status of SNTP.

[Syntax]
rtn = statsntp() ;

[Parameters]
No parameters

[Return Value]
rtn Returns one of the following values based on the status of SNTP:
0: Not started
(If **statsntp** function is called when time is not retrieved from the SNTP server)
1: Success
2: In progress
(this occurs when you call up the **statsntp** function while the **getsntptime** function is still in the progress of getting time from the SNTP server)
Note: It may take some time to search the server.
a value less than 0: Failure

[Example]

```
rtn = statsntp() ;          // rtn: the status of SNTP
```

13.3 String Manipulation Functions

13.3.1 chr() Convert 1 Byte Numerical Value to 1 Byte Character Code

[Description]
Converts a 1 byte numerical value to 1 byte character code.

[Syntax]
txtChar = chr(i) ;

[Parameters]

i	Numerical value

[Return Value]

txtChar	Success: returns the converted character code

[Example]

```
txtChar = chr(0x31) ;          //txtChar: '1'
```

13.3.2 asc() Convert Character Code of a Character to Numerical Value

[Description]
Converts the character code of a leading character of string to a numerical value.

[Syntax]
val = asc(txtStr) ;

[Parameters]

txtStr	The text to be converted (txtStr[0] is the only target of this function)

[Return Value]

val	Success: returns the converted numerical value

[Example]

```
val = asc(“1234”) ;        //val:0x31 The first character, 1, is the only target of this
                           function
```

13.3.3 **string()** List Specified Number of Specified Characters

[Description]
Creates a string with a specified number of specified characters.

[Syntax]
nStr = string(cnt,txtChar) ;

[Parameters]

cnt	the number of occurrences
txtChar	the characters to be listed

[Return Value]

nStr	Success: returns the created string
	Failure: returns a NULL string

[Example]

```
nStr = string(5,"a") ;          //nStr: "aaaaa"
```

13.3.4 **trim()** Remove Spaces in Front/Subsequent of String

[Description]

Creates a string by removing spaces in front and subsequent of a string.

Note: If a numerical value is specified, a numerical string will be returned.

[Syntax]

trimStr = trim(str) ;

[Parameters]

str	A string

[Return Value]

trimStr	Success: returns a string whose spaces in front and subsequent of it have been removed **Note**: If a numerical value (integer or floating point) is specified, a numerical string will be returned.

[Example]

1. Specifying a String

```
trimStr = trim(“  Hello!!!   “) ;          //triStr: “Hello!!!”
```

2. Specifying a Numerical Value

```
trimStr = trim(136)      ;          //triStr: “136”
trimStr = trim(10.56) ;             //triStr: “10.560000”
```

13.3.5 **len()** Get String Length

[Description]

Gets the length of a string.

Note: If a numerical value is specified, the returned value will be the length of a string converted from the numerical value.

[Syntax]

lval = len(str) ;

[Parameters]

str	A string

[Return Value]

lval	Success: returns the length of the string (the value will be equal to or larger than zero)

Note: If a numerical value (integer or floating point) is specified, the length of a numerical string will be returned.

[Example]

1. Specifying a String

```
lval = len(“Hello!!!”) ;          // lval: 8
```

2. Specifying a Numerical Value

```
lval = len(136) ;                 // lval:3 (length of string “136”)
lval = len(10.56) ;               // lval:9 (length of string “10.560000”)
```

13.3.6 **strcmp()** Compare Strings (with Range Specification)

[Description]

Compares two strings.

Compares up to the point specified in the range parameter, if the parameter is specified.

[Syntax]

val = strcmp(str1,str2[,n]) ; **Note**: Values enclosed in [ ] are optional.

[Parameters]

str1	Comparison string 1
str2	Comparison string 2
n	Comparison range of the string (optional)

[Return Value]

val	Success: returns one of the following values accordingly.
	0 : Matched
	A value of less than 0: str1 is smaller than str2.
	Note: returns –1, if the string range is specified.
	A value of more than 0: str1 is larger than str2.
	Note: Returns 1, if the string range is specified.

[Example]

1. Range Not Specified

```
val = strcmp(“Hello World !”,”Hello”) ;          // val: a value of more than 0
```

2. Range Specified

```
val = strcmp(“Hello World !”,”Hello”,5) ;        // val: 0
val = strcmp(“Hello World !”,”Helln”,5) ;        // val: 1
val = strcmp(“Hello World !”,”Hellp”,5) ;        // val: -1
```

[Remarks]

Strings are compared based on ASCII code.

13.3.7 **strchr()** Detect First Appearance of Specified Character

[Description]
Returns the position where the specified character appears for the first time in the string.

[Syntax]
val = strchr(str,txtChar) ;

[Parameters]

str	A string
txtChar	The character code for the search character

[Return Value]

val	Success: returns a value of 0 or more (position detected) Failure: returns -1 (position not found)

[Example]

```
val = strchr(“ABCDEFGH”,0x44) ;                    // val:
```

Note: The detected position assumes that the first character of the string is counted as “0.”

13.3.8 **strrchr()** Detect Last Appearance of Specified Character

[Description]
Returns the position where the specified character appears for the last time in the string.

[Syntax]
val = strrchr(str,txtChar) ;

[Parameters]

str	A string
txtChar	The character code for the search character

[Return Value]

val	Success: returns a value of 0 or more (position detected) Failure: returns -1 (the value not found)

[Example]

```
val = strrchr(“Hello World !”,0x6f) ;                    // val: 7
```

Note: The detected position assumes that the first character of the string is counted as “0.”

13.3.9 strstr() Detect First Appearance of Specified String

[Description]
Returns the position where the specified string appears for the first time in the string.

[Syntax]
val = strstr(str1,str2) ;

[Parameters]

str1	A string
str2	A search string

[Return Value]

val	Success: returns a value of 0 or more in relative number (position detected) Failure: returns -1 (the value not found)

[Example]

```
val = strstr(“Hello World !!!”,”World”) ;          // val:
```

Note: The detected position assumes that the first character of the string is counted as “0.”

13.3.10 **strmatch()** Compare Strings Using Wildcard

[Description]
Compares strings using wildcard(s).

[Syntax]
val = strmatch(str1,str2) ;

[Parameters]

str1	Comparison string
str2	A search string containing wildcard(s)

[Return Value]

val	Success: returns 1 (match found)
	Failure: returns 0 (no match found)

[Example]

1. Example 1
```
val = strmatch(“Hello World !!!”,”*W???d*”) ;          // val: 1
```

2. Example 2
```
val = strmatch(“Hello World !!!”,”*W*d*”) ;            // val: 1
```

3. Example 3
```
val = strmatch(“Hello World !!!”,”W*d”) ;              // val: 0
```

13.3.11 left() Extract String with Specified Number of Characters from Left

[Description]
Extracts the specified number of characters from left side of the string.

[Syntax]
lStr = left(str,n) ;

[Parameters]

str	A string
n	The length of string to be extracted

[Return Value]

lStr	Success: returns the extracted string
	Failure: returns 0 or NULL string

[Example]

```
lStr = left(“Hello World !!!”,5) ;          //lStr: “Hello”
```

Accordingly, str is assigned the values of 0x82, 0x50, and 0x82.

13.3.12 **right()** Extract String with Specified Number of Characters from Right

[Description]
Extracts the specified number of characters from right side of the string.

[Syntax]
rStr = right(str,n) ;

[Parameters]

str	A string
n	The length of string to be extracted

[Return Value]

rStr	Success: returns the extracted string
	Failure: returns 0 or NULL string

[Example]

```
rStr = right(“Hello World !!!”,3) ;          //rStr: “!!!”
```

13.3.13 **mid()** Extract Specified Number of Characters from Specified Position

[Description]

Extracts the specified number of characters from the specified position of the string.
If you do not specify the length, this function extracts to the leftmost character.

[Syntax]

mStr = mid(str,pos[,n]) ; **Note**: Values enclosed in [ ] are optional.

[Parameters]

str	A string
pos	The starting point of character extraction
n	The number of characters to be extracted (optional)

[Return Value]

mStr	Success: returns the extracted string
	Error: returns a NULL string

[Example]

1. Number Specified

```
mStr = mid(“Hello World !!!”,6,5) ;          //mStr: “World”
```

2. Number Not Specified

```
mStr = mid(“Hello World !!!”,6) ;            //mStr: “World
```

13.3.14 instr() Detect Position of Specified String from Specified Point

[Description]

Returns the position of the specified string in the text.
You can also specify the starting point of search and case-sensitivity.

[Syntax]

val = instr(str1,str2[,pos[,op]]) ; **Note**: Values enclosed in [ ] are optional.

[Parameters]

str1	A string
str2	A search string
pos	The starting position of search (optional; if not specified, search will be performed from the beginning of the string)
op	1: Case-insensitive 0 (default): Case-sensitive

[Return Value]

val	Success: returns a value of 0 or more (position detected) Failure: returns -1 (position not found)

[Example]

1. Starting Position Not Specified (Case-Sensitive)

```
val = instr(“Hello World !!!”,”World”);                    // val: 6
```

2. Starting Position Specified (Case-Sensitive)

```
val = instr(“Hello World !!!”,”World”,4);                  // val: 6
```

3. Starting Position Specified (Case-Insensitive)

```
val = instr(“Hello World !!!”,”world”,4,1);                // val: 6
```

13.3.15 **tolower()** Convert Alphabet String to Lower Case

[Description]
Converts upper case alphabet in the string to lower case.

[Syntax]
chgStr = tolower(str) ;

[Parameters]
str A string

[Return Value]
chgStr Success: returns the converted string

[Example]

```
chgStr = tolower(“Hello World !!!”) ;          // chfStr: “hello world !!!”
```

13.3.16 **toupper()** Convert Alphabet String to Upper Case

[Description]
Converts lower case alphabet in the string to upper case.

[Syntax]
chgStr = toupper(str) ;

[Parameters]
str A string

[Return Value]
chgStr Success: returns the converted string

[Example]

```
chgStr = toupper(“Hello World !!!”) ;          //chfStr: “HELLO WORLD !!!”
```

13.4 Hash Functions

13.4.1 gethashcount() Get Count of Hash Items

[Description]
Gets the number of items in an associative array.

[Syntax]
itemNo = gethashcount(hash[ ]) ;

[Parameters]
hash[ ] An associative array

[Return Value]
itemNo Success: returns a value of 0 or more (indicating the number of items)

[Example]

1. Example 1

```
var hash[ ] = {0xa4,0xa2,0xa4,0xa4,0xa4,0xa6,0xa4,0xa8,0xa4,0xaa} ;
itemNo = gethashcount(hash) ;                    // itemNo: 10
```

2. Example 2

```
t[“year”] = 2004 ;
t[“month”] = 1 ;
t[“day”] = 26 ;
t[“hour”] = 14 ;
t[“minute”] = 35 ;
t[“second”] = 20 ;
itemNo = gethashcount(t) ;                       // itemNo: 6
```

13.4.2 **sethashcount()** Set Number of Hash Elements

[Description]
Sets the number of items in an associative array.

[Syntax]
sethashcount(&hash[ ],itemNo) ;

[Parameters]

hash[ ]	An associative array
itemNo	The number of items

[Return Value]
No return value

[Example]

1. Example 1

```
var hash[ ] = {0xa4,0xa2,0xa4,0xa4,0xa4,0xa6,0xa4,0xa8,0xa4,0xaa} ;
sethashcount(&hash[ ],6) ;
// decrease the number of hash elements from 10 to 6
```

Note: Any data stored in seventh elements and later are cleared.

2. Example 2

```
sethashcount(&hash[ ],20) ;
// increase the number of hash elements from 10 to 20
```

13.4.3 **gethashkey()** Get Key Name at Specified Position in Hash

[Description]
Gets the key name at the specified position in an associative array.

[Syntax]
keyStr = gethashkey(hash[ ],n) ;

[Parameters]

hash[ ]	An associative array
n	The index number (relative number) of the position to get its key name

[Return Value]

keyStr	Success: returns a key name string Failure: returns a NULL string (“ “)

[Example]

```
t[“year”] = 2004 ;
t[“month”] = 1 ;
t[“day”] = 26 ;
t[“hour”] = 14 ;
t[“minute”] = 35 ;
t[“second”] = 20 ;
keyStr = gethashkey(t,3) ;          // keyStr: “hour
```

13.4.4 **sethashkey()** Set Key Name to Specified Position in Hash

[Description]
Sets a key name at the specified position in an associative array.

[Syntax]
sethashkey(&hash[ ],n,keyStr) ;

[Parameters]

hash[ ]	An associative array
n	The index number to set (in relative number)
keyStr	The key name string to set

Note: Any special characters [such as %, \, “,] cannot be used alone.

[Return Value]
No return value

[Example]

```
t[“year”] = 2004 ;
t[“month”] = 1 ;
t[“day”] = 26 ;
t[“hour”] = 14 ;
t[“minute”] = 35 ;
t[“second”] = 20 ;
sethashkey(&t[ ],6,”millisec”) ;
        // set “millisec” to the index number 6 as its key name.
```

13.4.5 **ishashkey()** Check Existence of Hash Key

[Description]
Checks if the specified key name exists in the associative array.

[Syntax]
pos = ishashkey(hash[ ],keyStr) ;

[Parameters]

hash[ ]	An associative array
keyStr	A key name string

[Return Value]

pos	returns a value of 0 or more: the index number of the specified key name returns -1: the specified key name does not exist

[Example]

```
t[“year”] = 2004 ;
t[“month”] = 1 ;
t[“day”] = 26 ;
t[“hour”] = 14 ;
t[“minute”] = 35 ;
t[“second”] = 20 ;
pos = ishashkey(t[ ],”hour”) ;                    // pos: 3
```

13.4.6 **hashinsert()** Insert Hash Item

[Description]
Inserts a hash item right before the specified position in an associative array.

[Syntax]
hashinsert(&hash[ ],n,keyVal) ;

[Parameters]

hash[ ]	An associative array
n	The index number of the insertion point (in relative number) or the key name (string)
keyVal	A value to be inserted

[Return Value]
No return value

[Example]

1. Value Key

```
hashinsert(&hash[ ],5,10) ;          // hash[5]: 10
```

2. String Key

```
t[“year”] = 2004 ;
t[“month”] = 1 ;
t[“day”] = 26 ;
t[“hour”] = 14 ;
t[“minute”] = 35 ;
t[“second”] = 20 ;
hashinsert(&t[ ],”day”,12) ;          // insert 12 in front of t[“day”]
```

13.4.7 **hashdelete()** Delete Hash Item

[Description]
Deletes an item in an associative array.

[Syntax]
hashdelete(&hash[ ],n) ;

[Parameters]

hash[ ]	An associative array
n	The number (relative number) or the key name (string) of an item to be deleted

[Return Value]
No return value

[Example]

1. Value Key

```
hashdelete(&hash[ ],3) ;                // delete the hash[3] item
```

2. String Key

```
t[“year”] = 2004 ;
t[“month”] = 1 ;
t[“day”] = 26 ;
t[“hour”] = 14 ;
t[“minute”] = 35 ;
t[“second”] = 20 ;
hashdelete(&t[ ],” minute”) ;           // t[“minute”] will be deleted
```

13.4.8 **hashfind()** Find Hash Contents

[Description]
Searches an associative array and returns the index of the specified value.

[Syntax]
keyNo = hashfind(hash[ ],keyVal) ;

[Parameters]

hash[ ]	An associative array
keyVal	A value to look for

[Return Value]

keyNo	Success: returns a value of 0 or more (indicating the index of the specified value) Failure: returns -1 (the value not found)

[Example]

```
t[“year”] = 2004 ;
t[“month”] = 1 ;
t[“day”] = 26 ;
t[“hour”] = 14 ;
t[“minute”] = 35 ;
t[“second”] = 20 ;
keyNo = hashfind(t,26) ;                    // keyNo: 2
```

13.4.9 **hashkeysort()** Sort by Hash Key

[Description]
Sorts an associative array by key.

[Syntax]
hashkeysort(&hash[ ],order,icase) ;

[Parameters]

hash[ ]	An associative array to be sorted	
order	The sorting order	
	0: Ascending,	1: Descending
icase	Case sensitivity	
	0: Case-sensitive	1: Case-insensitive

[Return Value]
No return value

[Example]

```
dat [“a”] = 10 ;
dat [“d”] = 40 ;
dat [“e”] = 50 ;
dat [“b”] = 20 ;
dat [“c”] = 30 ;
hashkeysort(&dat[ ],0,0) ;          // sort ascending by key
                                    // before sorting — 0:dat[“a”],1:dat [“d”] - `4:dat[“c”]
                                    // after sorting — 0:dat[“a”],1:dat [“b”] - 4:dat[“e”]
```

13.4.10 **hashsort()** Sort by Hash Contents

[Description]
Sorts an associative array by the contents.

[Syntax]
hashsort(&hash[ ],order,icase) ;

[Parameters]

hash[ ]	An associative array to be sorted	
order	The sorting order	
	0: Ascending,	1: Descending
icase	Case sensitivity	
	0: Case-sensitive	1: Case-insensitive

[Return Value]
No return value

[Example]

```
dat [“a”] = 10 ;
dat [“d”] = 40 ;
dat [“e”] = 50 ;
dat [“b”] = 20 ;
dat [“c”] = 30 ;
hashkeysort(&dat[ ],0,0) ;          // sort an associative array in ascending order by value
                                    // before sorting — 0:dat[“a”],1:dat [“d”] - `4:dat[“c”]
                                    // after sorting — 0:dat[“a”],1:dat [“b”] - 4:dat[“e”]
```

13.4.11 **hashcopy()** Copy Hash

[Description]
Copies the contents of an associative array.

[Syntax]
chash[ ] = hashcopy(hash[ ],start[,len]) ; **Note:** Values enclosed in [ ] are optional.

[Parameters]

hash[ ]	The source associative array.
start	The position to start copying
len	The length of data to be copied. If not specified, all contents to the end of the array will be copied.

[Return Value]

chash[ ]	returns the copied associative array

[Example]

1. Length Specified

```
dat [“a”] = 10 ;
dat [“d”] = 40 ;
dat [“e”] = 50 ;
dat [“b”] = 20 ;
dat [“c”] = 30 ;
chash[ ] = hashcopy(dat[ ],2,2);          // chash[0]=50, chash[1]=20
```

2. Length Not Specified

```
chash[ ] = hashcopy(dat[ ],2);            // chash[0]=50, chash[1]=20, chash[2]=30
```

13.4.12 **hashmerge()** Merge Hash Tables

[Description]
Merges multiple associative arrays into a single array.

[Syntax]
hash[ ] = hashmerge(hash1[ ][,hash2[ ],hash3[ ],…]) ; **Note**: Values enclosed in [ ] are optional.

[Parameters]
hash1[ ],hash2[ ],… Associative arrays to be merged
(specify at least two associative arrays)

[Return Value]
hash[ ] returns the merged associative array

[Example]

```
dat1[0][0] = 0 ;
dat1[0][1] = 1 ;
dat1[0][2] = 2 ;
dat2[1][0] = 20 ;
dat2[1][1] = 21 ;
dat2[1][2] = 22 ;
dat3[2][0] = 30 ;
dat3[2][1] = 31 ;
dat3[2][2] = 32 ;
mhash[ ] = hashmerge(dat1[ ][ ],dat2[ ][ ],dat3[ ][ ]) ;
            // mhash[0][0]=0, mhash[0][1]=1, mhash[0][2]=2,
            // mhash[1][0]=20, ... mhash[2][2]=32
```

13.4.13 **hashtostring()** Convert Hash to String

[Description]

Converts an associative array into a string

Note: It is assumed that the content of the associative array is an array of bytes.

[Syntax]

```
str = hashtostring(hash[ ]) ;
```

[Parameters]

hash[ ]	An associative array

[Return Value]

str	returns the converted string

[Example]

```
dats[0][0] = 0x41 ;
dats[0][1] = 0x42 ;
dats[0][2] = 0x43 ;
dats[0][3] = 0x44 ;
str = hashtostring(dats[ ][ ]) ;          // str: “ABCD”
```

13.4.14 **stringtohash()** Expand String to Hash

[Description]
Expands a string to an associative array.
The array will be an array of bytes.

[Syntax]
hash[ ] = stringtohash(str) ;

[Parameters]
str A string

[Return Value]
hash[ ] returns an array of bytes where the string is expanded

[Example]

```
hash[ ] = stringtohash(“ABCD”) ;
          // hash[0]=0x41, hash[1]=0x42, hash[2]=0x43, hash[3]=0x44
```

13.5. Regular Expression Functions

13.5.1 **regmatch()** Search String to Match Regular Expression

[Description]
Searches for text.
It terminates the search upon meeting the first string that matches the pattern.

[Syntax]
rtn = regmatch(“REG”,str,&hash[ ][,op]) ; **Note**: Values enclosed in [ ] are optional.

[Parameters]

“REG”	A regular expression pattern
str	A search string
hash[ ]	The array to store the result
op	Case sensitivity (optional) “i”: Case-insensitive Not Specified: Case-sensitive

[Return Value]

rtn	Success: returns 1 (a matching string found) hash[0]: returns the entire matching string hash[1 - n]: corresponds to the number of indices Failure: returns 0 (no matching string found)

[Example]

1. Code Sensitivity Not Specified (Case-Sensitive)
   rtn = regmatch(“.”,str,hex[ ]) ;

2. Case Sensitivity Specified (Case-Insensitive)
   rtn = regmatch(“.”,str,hex[ ],”i”) ;

[Remarks]
For more information on available regular expression patterns, see the list provided in “10.1 Regular Expression Operators.”

13.5.2 **regmatchall()** Search Strings to Match Regular Expression

[Description]
Searches text to extract all matching strings.

[Syntax]
cnt = regmatchall(“REG”, ,str,&hash[ ][,op]) ; **Note**: Values enclosed in [ ] are optional.

[Parameters]

“REG”	A regular expression pattern
str	A search string
hash[ ]	The array to store the result
op	Case sensitivity “i”: Case-insensitive Not Specified: Case-sensitive

[Return Value]

cnt	Success: returns a value of 0 or more (the number of matching strings)

[Example]

1. Code Sensitivity Not Specified (Case-Sensitive)
   cnt = regmatchall(“.”,str,hex[ ]) ;

2. Case Sensitivity Specified (Case-Insensitive)
   cnt = regmatchall(“.”,str,hex[ ],”i”) ;

[Remarks]
For more information on available regular expression patterns, see the list provided in “10.1 Regular Expression Operators.”

13.5.3 **regreplace()** Replace Strings with Regular Expression

[Description]
Performs text replacement.

[Syntax]
chgStr = regreplace(“REG”,toStr,bufStr[,op]) ; **Note**: Values enclosed in [ ] are optional.

[Parameters]

“REG”	A regular expression pattern
toStr	String to be replaced (\\1, \\2, \\3, ... and \\9 are placeholders with which the corresponding parenthesized group is replaced.)
bufStr	Text to be replaced
op	Case sensitivity “i”: Case-insensitive Not Specified: Case-sensitive

[Return Value]
chgStr Success: returns the replaced string

[Example]

1. Case Sensitivity Not Specified (Case-Sensitive)

```
var     testStr = “abcd” ;
chgStr = regreplace(“bc”, “12”, testStr) ;          // chgStr=”a12d”
```

2. Case Sensitivity Specified (Case-Insensitive)

```
var     testStr = “abcd” ;
chgStr = regreplace(“Bc”, “12”, testStr”i”) ;       // chgStr=”a12d”
```

3. Example of Using Parentheses

```
var     pattern =”(a)(b)” ;
var     testStr = “abcd” ;
chgStr = regreplace(pattern, “\\1x\\2y”, testStr) ;   // chgStr=”axbycd”
```

[Remarks]
For more information on available regular expression patterns, see the list provided in “10.1 Regular Expression Operators.”

13.6. File Manipulation Functions

13.6.1 **getcurdir()** Get Current Directory

[Description]
Gets the current directory.

[Syntax]
curpath = getcurdir() ;

[Parameters]
No parameters

[Return Value]

curpath	Success: returns the current path name
	Failure: returns a NULL string (“ “ means not set)

[Example]

```
curpath = getcurdir() ;          // curpath: string representing the current path name
```

13.6.2 **setcurdir()** Set Current Directory

[Description]
Sets the current directory.

[Syntax]
rtn = setcurdir(curpath) ;

[Parameters]

curpath	The directory name to set

Note: A directory path separator must be \\.
If the drive name is not provided, the directory will be set under the current directory.

[Return Value]

rtn	Success: returns 0
	Failure: returns -1

[Example]

1. Specifying the Full Path

```
rtn = setcurdir("B:\\test") ;      // set "B:\test" as the current directory
```

2. Specifying a Directory

```
rtn = setcurdir("test") ;          // set "test" directory under the current directory
                                   // as the new current directory
```

13.6.3 **dirmake()** Create Directory

[Description]
Creates a directory.

[Syntax]
rtn = dirmake(dirpath) ;

[Parameters]

dirpath	The directory path name to be created

Note: A directory path separator must be \\.
If a drive name is not provided, the directory will be created under the current directory.

[Return Value]

rtn	Success: returns 0
	Failure: returns -1

[Example]

1. Specifying the Full Path

```
rtn = dirmake("B:\\test") ;      // create "test" directory under B drive
```

2. Specifying a Directory

```
rtn = dirmake ("test") ;         // create "test" directory
                                 // under the current directory
```

13.6.4 **dirremove()** Remove Directory

[Description]
Removes a directory.

[Syntax]
rtn = dirremove(dirpath) ;

[Parameters]

dirpath	The directory path name

Note: A directory path separator must be \\.
If the drive name is not provided, the directory under the current directory will be removed.

[Return Value]

rtn	Success: returns 0
	Failure: returns -1

Note: This function cannot remove the directory, if any files exist in the specified directory.

[Example]

1. Specifying the Full Path
```
rtn = dirremove("B:\\test") ;          // delete the "test" directory under the B drive
```

2. Specifying a Directory
```
rtn = dirremove("test") ;              // delete the "test" directory
                                       // under the current directory
```

13.6.5 **diropen()** Open Directory

[Description]
Opens a directory.

[Syntax]
dirNo = diropen(dirpath) ;

[Parameters]

dirpath	The directory path name

Note: A directory path separator must be \\.
If the drive name is not provided, the directory under the current directory will be opened.

[Return Value]

dirNo	Success: returns a value of 0 or greater (representing the directory number) Failure: returns -1

[Example]

1. Specifying the Full Path
   dirNo = diropen(“B:\\”) ; // dirNo is equal to or greater than 0
2. Specifying a Directory
   dirNo = diropen(“test”) ; // dirNo is a equal to or greater than 0,
   // consequently the “test” directory under
   // the current directory will be opened.

13.6.6 **dirclose()** Close Directory

[Description]
Closes a directory.

[Syntax]
rtn = dirclose(dirNo) ;

[Parameters]

dirNo	The directory number
	Note: Specify the directory number returned by diropen() function.

[Return Value]

rtn	Success: returns 0
	Failure: returns -1

[Example]

```
rtn = dirclose(dirNo) ;
```

13.6.7 **dirread()** Get File Name from Directory

[Description]
Gets names of all files in a directory.

[Syntax]
fname = dirread(dirNo) ;

[Parameters]

dirNo	A directory number (a value of 0 or greater) **Note**: Specify the directory number returned by **diropen()** function.

[Return Value]

fname	Success: returns string(s) representing file name(s) Error: returns a NULL string

[Example]

```
fname = dirread(dirNo) ;          // get all file names in a directory
```

13.6.8 **filecreat()** Create File

[Description]
Creates a file.

[Syntax]
fd = filecreat(fpath) ;

[Parameters]

fpath	The file path name

Note: A file path separators must be \\.
If no drive name is provided, the file will be created under the current directory.

[Return Value]

fd	Success: returns a file descriptor (equal to or more than 0) Failure: returns -1

[Example]

1. Specifying the Full Path

```
fd = filecreat("B:\\test.txt");          // create a files named "test.txt" under the B drive
```

2. Specifying a File Name

```
fd = filecreat("test.txt") ;             // create a file named "test.txt"
                                         // under the current directory
```

Note: The filecreat() function is write-only.
To load the created file, you need to close the file and then re-open or you will get an error if you try to read the created file straight away.

13.6.9 **fileopen()** Open File

[Description]
Opens a file.

[Syntax]
fd = fileopen(fpath,mode) ;

[Parameters]

fpath	The file path name **Note**: A file path separators must be \\. If no drive name is provided, the file under the current directory will be opened
mode	Access mode (use one of the following value as appropriate) “r”: Read mode “w”: Write mode “a”: Append mode “rw”: Read/Write mode

[Return Value]

fd	Success: returns a value of 0 or greater (indicating a file descriptor) Failure: returns -1

[Example]

```
fd = fileopen(“B:\\test.txt”,”rw”) ;          // open test.txt file under the B drive
```

13.6.10 **fileclose()** Close File

[Description]
Closes a file.

[Syntax]
rtn = fileclose(fd) ;

[Parameters]

fd	A file descriptor
	Note: Specify the file descriptor returned by **filecreat()** or **fileopen()** function.

[Return Value]

rtn	Success: returns 0
	Failure: returns -1

[Example]

```
rtn = fileclose(fd) ;
```

Note: If the parameter passed on to fileclose function (fd) is –1, script terminates its process immediately. The return value of the fileopen function must be checked prior to calling the fileclose function.

13.6.11 **fileread()** Read Strings from File

[Description]
Reads string data from a file.

[Syntax]
rtn = fileread(fd,buf,readCnt) ;

[Parameters]

fd	A file descriptor
	Note: Specify the file descriptor returned by **fileopen()** function.
buf	The area for storing read data
readCnt	The size of data to be read
	Note: If no file size is provided, this function will load the whole data in the file. If the provided size or the file size exceeds 4,096 bytes, this function will read only up to 4,096 bytes of data.

[Return Value]

rtn	Success: returns a value of 0 or greater (indicating the size of read data)
	Failure: returns -1

[Example]

```
rtn = fileread(fd,buf,128) ;
```

13.6.12 **filewrite()** Write String to File

[Description]
Writes string data to a file.

[Syntax]
rtn = filewrite(fd,buf,writeCnt) ;

[Parameters]

fd	A file descriptor
	Note: Specify the file descriptor returned by filecreat() or fileopen() function.
buf	The area for storing data to be written
writeCnt	The size of data to be written
	Note: If no data size is provided, this function will write all string data into the file. However, if the specified size or the file size exceeds 4,096 bytes, this function will write only up to 4,096 bytes of data.

[Return Value]

rtn	Success: returns a value of 0 or greater (indicating the size of written data)
	Failure: returns -1

[Example]

```
rtn = filewrite(fd,buf,128) ;
```

13.6.13 **filereadhash()** Load Data from File into Associative Array

[Description]
Reads the specified number of data bytes from a file and loads it into an associative array. This function is capable of loading up to 4,096 data bytes at a time.

[Syntax]
```
rtn = filereadhash(fd,&buf[ ],len) ;
```

[Parameters]

fd	A file descriptor
	Note: Specify the file descriptor returned by filereadhash() or fileopen() function.
buf[ ]	An associative array
len	The number of data bytes to be read
	Note: If no file size is provided at the time of loading the data, this function will try to load all the data in the file. However, if the provided size or the file size exceeds 4,096 bytes, this function will read only up to 4,096 bytes of data.

[Return Value]

rtn	Success: returns a value of 0 or greater (indicating the number of data bytes read)
	Failure: returns -1

[Example]

1. Specifying the Number of Data Bytes to be Read
```
rtn = filereadhash(fd,&buf[ ],128) ;          // rtn is equal to or less than 128
```

2. Not Specifying the Number of Data Bytes to be Read
```
rtn = filereadhash(fd,&buf[ ]) ;              // rtn is equal to or less than 4,096
```

13.6.14 **filewritehash()** Write Associative Array Data into File

[Description]
Writes specified number of bytes data from an associative array into a file.
This function is capable of writing up to 4.096 data bytes at a time.

[Syntax]
rtn = filewritehash(fd,buf[ ],len) ;

[Parameters]

fd	A file descriptor **Note**: Specify the file descriptor returned by **filereadhash()** or **fileopen()** function.
buf[ ]	An associative array
len	The number of data bytes to be written **Note**: If "0" is provided at the time of writing data, this function will try to write out all data contained in the associative array. However, if the specified size or the file size exceeds 4,096 bytes, this function will only write up to 4,096 bytes of data.

[Return Value]

rtn	Success: returns a value of 0 or greater (indicating the number of data bytes written) Failure: returns -1

[Example]

1. Specifying the Number of Data Bytes to be Written
```
rtn = filewritehash(fd,&buf[ ],128) ;          // rtn is equal to or less than 128
```

2. Not Specifying the Number of Data Bytes to be Written
```
rtn = filewritehash(fd,&buf[ ]) ;              // rtn is equal to or less than 4,096
```

13.6.15 **getfilesize()** Get File Size

[Description]
Gets the file size.

[Syntax]
rtn = getfilesize(fd) ;

[Parameters]

fd	A file descriptor **Note**: Specify the file descriptor returned by **fileopen()** function.

[Return Value]

rtn	Success: returns a value of 0 or greater (indicating the file size) Failure: returns -1

[Example]

```
rtn = getfilesize(fd) ;
```

13.6.16 **filedelete()** Delete File

[Description]
Deletes a file.

[Syntax]
rtn = filedelete(fpath) ;

[Parameters]

fpath	The file path name

Note: A file path separators must be \\.
If no drive name is provided, the file under the current directory will be deleted.

[Return Value]

rtn	Success: returns 0
	Failure: returns -1

[Example]

1. Specifying the Full Path

```
rtn = filedelete("B:\\test.txt") ;        // delete the file named test.txt under the
                                             B drive
```

2. Specifying a File Name

```
rtn = filedelete("test.txt") ;            // delete the file named "test.txt"
                                          // under the current directory
```

Note: The filedelete function requires you to have write privileges ("w") for the file to be deleted.

13.6.17 **filerename()** Rename File

[Description]
Renames a file.

[Syntax]
rtn = filerename(fpath,newfname) ;

[Parameters]

fpath	The target file path name **Note**: A file path separators must be \\. If no drive name is provided, the file under the current directory will be renamed.
newfname	The new file name **Note**: For the new file name, specify only the file name.

[Return Value]

rtn	Success: returns 0 Failure: returns -1

[Example]

1. Specifying the Full Path of the Target File

```
rtn = filerename("B:\\test.txt","abc.txt") ;      // rename the "test.txt" file
                                                  // under the current directory
```

2. Specifying the Target File Name

```
rtn = filerename("test.txt","abc.txt") ;          // rename the test.txt file
                                                  // under the current directory
```

13.6.18 **filecopy()** Copy File

[Description]
Copies a file.

[Syntax]
rtn = filecopy(srcfname,dstfname) ;

[Parameters]

srcfname	The source file path name
dstfname	The target file path name

Note: A file path separators must be \\.
If no drive name is provided, a file under the current directory is assumed to be the target of this process.

[Return Value]

rtn	Success: returns 0
	Failure: returns -1

[Example]

1. Specifying the Full Path

```
// copy the test.txt file in the B drive to the C drive as a file named “abc.txt”
rtn = filecopy(“B:\\test.txt”,”C:\\abc.txt”) ;
```

2. Specifying a File Name

```
// copy the “text.txy” file under // the current directory to the C drive
// as a file named “abc.txt”
rtn = filecopy(“test.txt”,”C:\\abc.txt”) ;
```

Note: The filecopy function requires you to have write privileges (“w”) for the file to be copied.

13.6.19 **filetell()** Get Access Position in File

[Description]
Gets the access position in a file.

[Syntax]
rtn = filetell(fd) ;

[Parameters]

fd	A file descriptor **Note**: Specify the file descriptor returned by **filecreat()** or **fileopen()** function.

[Return Value]

rtn	Success: returns a value of 0 or greater (indicating the access position) Failure: returns -1

[Example]

```
rtn = filetell(fd) ;          // rtn=the access position in a file
```

13.6.20 **filelseek()** Move Access Position in File

[Description]
Moves the access position in a file.

[Syntax]
rtn = filelseek(fd,offset,base) ;

[Parameters]

fd	A file descriptor **Note**: Specify the file descriptor returned by **filecreat()** or **fileopen()** function.
offset	Travel distance from the base position
base	The base position in a file (use one of the following values as appropriate) 0: From the beginning of a file 1: From the current position 2: From the end of a file

[Return Value]

rtn	The access position after going through the process of this function Failure: returns -1

If the provided size is greater than the size of the file you opened, one of the following values will be returned:

When opened the file in “r” mode: returns -1
When opened the file in “rw” or “w” mode: returns the sought position

[Example]

```
rtn = filelseek(fd,100,0) ;          // move 100 bytes away from the beginning of the file
```

13.6.21 **filechmod()** Change File Attributes

[Description]
Changes the mode attributes of a file.

[Syntax]
rtn = filechmod(fpath,mode) ;

[Parameters]

fpath	The file path name **Note:** A file path separators must be \\. If no drive name is provided, a file under the current directory is assumed to be the target of this process.
mode	File attributes: “r”: Read-only “w”: Write-only “rw”: Read/Write

[Return Value]

rtn	Success: returns 0 Failure: returns -1

[Example]

1. Specifying the Full Path

```
rtn = filechmod(“B:\\test.txt”,”rw”) ;        // set the mode attribute of the file
                                              // to Read/Write
```

2. Specifying a File Name

```
rtn = filechmod(“test.txt”,”rw”) ;            // set the mode attribute of
                                              // “test.txt” file under the current
                                              //  directory to Read/Write
```

13.6.22 **fputs()** Output One Line String

[Description]

Writes a line of string data to a file.

If no new line character (“\r\n”) exists at the end of the line, a new line character will be appended at the time of output.

[Syntax]

```
rtn = fputs(fd,str) ;
```

[Parameters]

fd	A file descriptor
	Note: Specify the file descriptor returned by **filecreat()** or **fileopen()** function.
str	String data

[Return Value]

rtn	Success: returns 0
	Failure: returns -1

[Example]

1. Example 1

```
rtn = fputs(fd,”It’s fine today”) ;          // output “It’s fine today\r\n”
```

2. Example 2

```
rtn = fputs(fd,”It’s fine today\n”) ;        // output “It’s fine today\r\n”
```

3. Example 3

```
rtn = fputs(fd,”It’s fine today\r\n”) ;      // output “It’s fine today\r\n”
```

13.6.23 **fgets()** Get One Line String

[Description]
Gets a line of string data from a file.

[Syntax]
rtn = fgets(fd,str,maxline) ;

[Parameters]

fd	A file descriptor **Note**: Specify the file descriptor returned by **fileopen()** function.
str	The area for storing the read data
maxline	The maximum data size of a line (the maximum size is 256 bytes)

[Return Value]

rtn	Success: returns a value of 0 or greater (indicating the length of string) Failure: returns -1 **Note**: In case of an error, the file pointer loses its position and you will need to re-specify the appropriate position of the file.

[Example]

```
rtn = fgets(fd,str,255) ;          //str: loaded_string
```

Note: The fgets function stops loading the string when meeting any of the following characters: “\r”, “\n”, “\r\n”, “\0”, or “EOF.” The loaded character string does not include any new line code.

13.6.24 **fflush()** Refresh the flash file (C:\) drive.

[Description]
Refresh the flash file (C:\) drive.

[Syntax]
rtn = fflush() ;

[Parameters]
No parameters

[Return Value]
rtn Success: returns a value of 0
Failure: returns -1

[Example]

```
rtn = fflush() ;          //Refresh the flash file (C:\) drive.
```

13.7 I/O Functions

13.7.1 puts() Output String to Standard Output

[Description]
Outputs a string to standard output.

[Syntax]
puts(str) ;

[Parameters]
str A string

[Return Value]
No return value

[Example]

```
puts(“It’s fine today”) ;
```

13.7.2 **gets()** Get String from Standard Input

[Description]
Gets a string from standard input.

[Syntax]
str = gets(n) ;

[Parameters]

n	The length of characters to be obtained (the maximum of 1,024 characters)

[Return Value]

str	returns the obtained string

[Example]

```
str = gets(16) ;          // get 16 characters
```

13.7.3 **getchar()** Get Character from Standard Input

[Description]
Gets a character from standard input.

[Syntax]
str = getchar() ;

[Parameters]
No parameters

[Return Value]
str Success: returns the obtained character

[Example]

```
str = getchar() ;          // get one character
```

13.7.4 **beep_on ()** Turn Beep on

[Description]
Beeps.

[Syntax]
rtn = beep_on (hz,msec) ;

[Parameters]

hz	Frequency (10 - 10000 Hz)
msec	Beep duration (in msec) 0: continuance 1 - 1000: beep duration of your choice **Note**: Other processes may be stopped when the beep is turned on.

[Return Value]

rtn	Success: returns 0 Failure: returns -1

[Example]

```
rtn = beep_on (100,10) ;          // beeps for 10 msec at the frequency of 100 Hz
```

13.7.5 **beep_off ()** Turn Beep off

[Description]
Stops the beep.

[Syntax]
rtn = beep_off () ;

[Parameters]
No parameters

[Return Value]

rtn	Success: returns 0
	Failure: returns -1

[Example]

```
rtn = beep_off () ;          // turn the beep off
```

13.8. SIO I/O Functions

13.8.1 **sioputs()** Output String to Serial

[Description]
Outputs a string to a serial port.

[Syntax]
rtn = sioputs(ch,str[,tmout]) ; **Note**: Values enclosed in [ ] are optional.

[Parameters]

ch	Channel number (0 - 3)
str	The output string
tmout	Timeout (20 - 1000 in msec) **Note**: Defaults to 1,000 msec if not specified.

[Return Value]

rtn	Success: returns a value of 0 or more (indicating number of output characters) Failure: returns a value of less than 0 **Note**: For more information about error codes, see “Detailed Information about Error Codes” on page 234

[Example]

1. Timeout Not Specified (default timeout)

```
rtn = sioputs(0,”Hello!”) ;          // output string to channel 0
                                     // timeout after 1,000 msec
```

2. Timeout Specified

```
rtn = sioputs(0,”Hello!”,50) ;       // output string to channel 0
                                     // timeout after: 50 msec
```

13.8.2 **siogets()** Get Data String from Serial

[Description]
Gets string data from serial port input data.

[Syntax]
str = siogets(ch,len[,tmout]) ; **Note**: The value enclosed in [ ] is optional.

[Parameters]

ch	Channel number (0 - 3)
len	The length of string to be obtained
tmout	Timeout (20 - 1000 in msec)
	Note: Defaults to 1,000 msec if not specified.

[Return Value]

str	Success: returns obtained string

[Example]

1. Timeout Not Specified (default timeout)

```
str = siogets(0,8) ;          // get 8 characters from channel 0
```

2. Timeout Specified

```
str = siogets(0,8,500) ;      // get 8 characters from channel 0
                              // timeout after: 50 msec
```

Notes: New line characters will be ignored.

siogets function stops loading the string when meeting any of the following characters: “\r”, “\n”, “\r\n”, “\0”, or “EOF.”
The loaded character string does not include any new line code.

13.8.3 **sioputb()** Output Binary Data to Serial

[Description]
Outputs one byte of binary data to a serial port.

[Syntax]
rtn = sioputb(ch,outDat[,tmout]) ; **Note**: The value enclosed in [ ] is optional.

[Parameters]

ch	Channel number (0 - 3)
outDat	Binary data (1 byte)
tmout	Timeout (20 - 1000 in msec) **Note**: Defaults to 1,000 msec if not specified.

[Return Value]

rtn	Success: returns 0 Failure: returns -1 **Note**: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

1. Timeout Not Specified (default timeout)

```
rtn = sioputb(0,0x00) ;          // output 0x00 to channel 0
```

2. Timeout Specified

```
rtn = sioputb(0,0x00,300) ;      // output 0x00 to channel 0
                                 // timeout after: 300 msec
```

13.8.4 **siogetb()** Get Binary Data from Serial Port

[Description]
Gets one byte of binary data from a serial port.

[Syntax]
inDat = siogetb(ch[,tmout]) ; **Note**: Values enclosed in [ ] are optional.

[Parameters]

ch	Channel number (0 - 3)
tmout	Timeout (20 - 1000 in msec) **Note**: Defaults to 1,000 msec if not specified.

[Return Value]

inDat	Success: returns input binary data (1 byte) Failure: returns a value less than 0 **Note**: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

1. Timeout Not Specified (default timeout)

```
inDat = siogetb(0) ;              // input one byte from channel 0
```

2. Timeout Specified

```
inDat = siogetb(0,1000) ;         // input one byte from channel 0
                                  // timeout after: 1,000 msec
```

13.8.5 **sioctrl()** Set Serial Communication Speed

[Description]
Sets the communication speed of a serial port.

[Syntax]
rtn = sioctrl(ch,speed) ;

[Parameters]

ch	Channel number (0 - 3)
speed	The serial communication speed 600:600bps 1200: 1,200 bps 2400: 2,400 bps 4800: 4,800bps 9600: 9,600 bps 19200: 19,200 bps 38400: 38,400 bps 115200: 115,200 bps

[Return Value]

rtn	Success: returns 0 Failure: returns -1

[Example]

```
rtn = sioctrl(1,9600) ;                    // set speed of channel 1 at 9,600 bps
```

13.8.6 **siorxclr()** Clear Serial Receive Buffer

[Description]
Clears a serial receive buffer.

[Syntax]
rtn = siorxclr(ch) ;

[Parameters]

ch	Channel number (0 - 3)

[Return Value]

rtn	Success: returns 0 Failure: returns -1 **Note**: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

```
rtn = siorxclr(1) ;
```

13.8.7 **sioputd()** Output Binary Data Array to Serial

[Description]
Outputs a binary data array to a serial port.

[Syntax]
rtn = sioputd(ch,dat[],cnt[,tmout]) ; **Note**: Values enclosed in [ ] are optional.

[Parameters]

ch	Channel number (0 - 3)
dat[ ]	An associative array for storing output data
cnt	The count of items to be output
tmout	Timeout (20 - 1000 in msec) **Note**: Defaults to 1,000 msec if not specified.

[Return Value]

rtn	Success: returns a value of 0 or more (indicating the number of output items) Failure: returns a value less than 0 **Note**: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

1. Timeout Not Specified (default timeout)

```
var  dat[] = {0x01,0x33,0x00,0x00,0x04} ;
rtn = sioputd(0,dat[],5) ;              // output 5 bytes of data to channel 0
```

2. Timeout Specified

```
var  dat[] = {0x01,0x33,0x00,0x00,0x04} ;
rtn = sioputd(0,dat[],5,300) ;          // output 5 bytes of data to channel 0
                                        // timeout after: 300 msec
```

13.8.8 **siogetd()** Get Binary Data Array from Serial Port

[Description]
Gets a binary data array from a serial port.

[Syntax]
len = siogetd(ch,&dat[],cnt[,tmout]) ; **Note**: Values enclosed in [ ] are optional.

[Parameters]

ch	Channel number (0 - 3)
dat[ ]	Input data storage
cnt	The requesting number of items to be retrieved
tmout	Timeout (20 - 1000 in msec) **Note**: Defaults to 1,000 msec if not specified.

[Return Value]

rtn	Success: returns a value of 0 or more (indicating the number of retrieved items) Failure: returns a value less than 0 **Note**: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

1. Timeout Not Specified (default timeout)

```
var  dat[] ;
len = siogetd(0,&dat[],5) ;              // get 5 bytes of data from channel 0
```

2. Timeout Specified

```
var  dat[] ;
len = siogetd(0,&dat[],5,1000) ;         // get 5 bytes of data from channel 0
                                         // timeout after: 1,000 msec
```

13.8.9 **siotxclr()** Clear Serial Transmit Buffer

[Description]
Clears a serial transmit buffer.

[Syntax]
rtn = siotxclr(ch) ;

[Parameters]

ch	Channel number (0 - 3)

[Return Value]

rtn	Success: returns 0 Failure: returns -1 **Note**: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

```
rtn = siotxclr(1) ;                    // clear the transmit buffer for channel 1
```

13.8.10 siosigctrl() Control the serial number string (RTS,DTR)

[Description]
Control the serial number string (RTS,DTR)

[Syntax]
rtn = siosigctrl(ch,sig,flg) ;

[Parameters]

ch	Channel number (0 - 3)
sig	Serial Number String (“RTS” or “DTR”)
flg	Control Value(0 or 1)

[Return Value]

rtn	Success: returns 0 Failure: returns -1 **Note**: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

```
rtn = siosigctrl(3,”RTS”,1) ;        //Output the RTS signal from COM3 to 1

rtn = siosigctrl(2,”DTR”,1) ;        // Output the DTR signal from COM2 to 1
```

13.8.11 getsiosig() Get the serial number string(CD,DSR,CTS)value.

[Description]
Get the serial number string(CD,DSR,CTS) value.
CD,DSR string values will only be valid if the signal string has been changed.

[Syntax]
rtn = getsiosig(ch,sig) ;

[Parameters]

ch	Channel number (0 - 3)
sig	Serial Number String (“CD” or “DSR” or “CTS“)

[Return Value]

rtn	Success: returns Signal value(0 or 1) Failure: returns -1

Note: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

```
rtn = getsiosig(3,”CD”) ;   // Get the CD signal value for COM3

rtn = getsiosig(2,”DSR”) ;// Get the DSR signal value for COM2

rtn = getsiosig(1,”CTS”) ; // Get the CTS signal value for COM1
```

13.8.12 com1_dtr_out() Control the COM1 DTR Signals

[Description]
Control the DTR signal from COM1.

[Syntax]
com1_dtr_out(dtr);

[Parameters]
dtr The DTR signal output value (0 or 1)

[Return Value]
No return value

[Example]

1. DTR output

```
com1_dtr_out(0);                    // set the DTR signal output for COM1 to 0
```

13.8.13 COM1_ptn_in()

[Description]
Controls DTR, DSR and DCD signal line inputs for COM1.

[Syntax]
ptnDat=com1_ptn_in();

[Parameters]
No parameters

[Return Value]
ptnDat The input bit value of DTR, DSR and DCD for COM1
Formatted as a 8-bit binary pattern (00000xyz) where:
x: DCD signal value (0 or 1)
y: DSR signal value (0 or 1)
z: DTR signal value (0 or 1)

[Example]

1. DTR, DSR and DCD signal line inputs for COM1

```
ptnDat=com1_ptn_in();                    // input each signal line for COM1
```

13.8.14 com1_ctrl()

[Description]
Controls RTS and CTS for COM1.

[Syntax]
cts = com1_ctrl(rts);

[Parameters]

rts	The RTS signal control value 0: Off 1: On

[Return Value]

cts	The CTS signal input value (0 or 1)

[Example]

```
COM1 control test
    cts = com1_ctrl(1);
```

13.9 PIO I/O Functions

13.9.1 **putport()** Output 1 Bit Data to PIO Port

[Description]
Outputs 1-bit data to the specified PIO port.

[Syntax]
rtn = putport(portNo,outDat) ;

[Parameters]

portNo	A port number (0 -15)
outDat	The output bit data (0 or 1)

[Return Value]

rtn	Success: returns 0 Failure: returns -1 **Note**: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

```
rtn = putport(2,1) ;                    // output 1 to port 2
```

13.9.2 **getport()** Get 1 Bit Data from PIO Port

[Description]
Gets 1-bit data from the specified PIO port.

[Syntax]
inDat = getport(portNo) ;

[Parameters]

portNo	A port number (0 -15)

[Return Value]

inDat	Success: returns 1-bit data retrieved (0 or 1) Failure: returns any value other than 0 and 1

Note: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

```
inDat = getport(2) ;
```

13.9.3 **putpio()** Output 1 Bit Data to Multiple PIO Ports

[Description]
Outputs 1-bit data to multiple PIO ports from the corresponding port.

[Syntax]
putpio(pioBits,outBits) ;

[Parameters]

pioBits	The bit strings indicating the port number (0x0000 - 0xFFFF) Higher-order bit (Port number 15) <--- Lower-order bit (Port number 0)
outBits	The output bit string data corresponding to the port number bit string (0x0000 - 0xFFFF)

[Return Value]
No return value

[Example]

```
putpio(0xAA55,0x2211) ;
```

13.9.4 **getpio()** Get 1 Bit Data from Multiple PIO Ports

[Description]
Gets the bit data of the corresponding port from multiple PIO ports.

[Syntax]
inBits = getpio(pioBits) ;

[Parameters]

pioBits	The bit strings indicating the port number (0x0000 - 0xFFFF) Higher-order bit (Port number 15) <--- Lower-order bit (Port number 0)

[Return Value]

inBits	Success: returns retrieved bit corresponding data Failure: returns a value less than 0 Note: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

```
inBits =  getpio(0xA5A5) ;
```

13.9.5 **setport()** Set I/O Direction of PIO Port

[Description]
Sets the I/O direction of the specified PIO port.

[Syntax]
rtn = setport(portNo,dir) ;

[Parameters]

portNo	The PIO port number (0 -15)
dir	The I/O direction (0: Input, 1: Output)

[Return Value]

rtn	Success: returns 0 Failure: returns a value less than 0 **Note**: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

```
rtn = setport(1,1) ;                    // specify the direction of channel 1 to output
```

13.9.6 **setpio()** Set I/O Direction of Multiple PIO Ports

[Description]
Sets the I/O direction of multiple PIO ports.

[Syntax]
setpio(pioBits,dirBits) ;

[Parameters]

pioBits	The bit string indicating the port number (0x0000 - 0xFFFF) Higher-order bit (Port number 15) <--- Lower-order bit (Port number 0) **Note**: Specify the ports for which to set the I/O direction.
dirBits	The bit string indicating the I/O direction (0x0000 - 0xFFFF) The I/O direction (0: Input, 1: Output)

[Return Value]
No return value

[Example]

```
setpio(0xAA55,0x2211) ;
```

13.9.7 **getpiopulse()** Get PIO Port Pulse Count

[Description]
Gets the output pulse count of a PIO port.

[Syntax]
pulseNum = getpiopulse(portNo) ;

[Parameters]
portNo A port number (0 -15)

[Return Value]
pulseNum Success: returns a value of 0 or more
(indicating the number of output pulse)
Failure: returns a value less than 0
Note: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

```
pulseNum = getpiopulse(0) ;
```

Note: The pulse count is based on the occurrence of trailing.

13.9.8 **clrpiopulse()** Clear Count of PIO Port Pulse

[Description]
Clears the count of output pulse for a PIO port.

[Syntax]
rtn = clrpiopulse(portNo) ;

[Parameters]

portNo	A port number (0 -15)

[Return Value]

rtn	Success: returns 0 Failure: returns a value less than 0 **Note**: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

```
rtn = clrpiopulse (0) ;
```

Note: Use the **getpiopulse** function to count the number of pulse and, if needed, use the **clrpiopulse** function to clear the count.

13.9.9 **statport()** Get I/O Direction of PIO Port

[Description]
Gets the I/O direction of a PIO port.

[Syntax]
dir = statport(portNo) ;

[Parameters]

portNo	A port number (0 -15)

[Return Value]

dir	Success: returns the I/O direction (0: Input, 1: Output) Failure: returns a value less than 0 **Note**: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

```
dir = statport (0) ;
```

13.9.10 **statpio()** Get I/O Direction of Multiple PIO Ports

[Description]
Gets the I/O direction of multiple PIO ports.

[Syntax]
dirBits = statpio(pioBits) ;

[Parameters]

pioBits	The bit string indicating the port number (0x0000 - 0xFFFF) Higher-order bit (Port number 15) <--- Lower-order bit (Port number 0) **Note**: Specify the target ports.

[Return Value]

dirBits	Success: returns a bit string indicating the I/O direction (0x0000 - 0xFFFF) The I/O direction (0: Input, 1: Output) Failure: returns -1 :<0 **Note**: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

```
dirBits = statpio (0xFFFF) ;          // get the I/O direction of ports 0 - 15
```

13.9.11 **putexport()** Output 1 Bit Data to Extended Multipurpose PIO Port

[Description]
Outputs 1-bit data to the specified extended multipurpose PIO port.

[Syntax]
rtn = putexport(portNo,outDat) ;

[Parameters]

portNo	The number of an extended PIO port (0 - 7) **Note**: Port number 4 - 7 are not available as the output target.
outDat	The output bit data (0 or 1)

[Return Value]

rtn	Success: returns 0 Failure: returns a value less than 0 **Note**: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

```
rtn = putexport(2,1) ;          // output 1 to port 2
```

13.9.12 **getexport()** Get 1 Bit Data from Extended Multipurpose PIO Port

[Description]
Gets 1 bit data from the specified extended multipurpose PIO port.

[Syntax]
inDat = getexport(portNo) ;

[Parameters]

portNo	The number of an extended PIO port (0 - 7)

[Return Value]

inDat	Success: returns 1-bit data retrieved (0 or 1) Failure: returns a value less than 0

Note: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

```
inDat = getexport(2) ;          // get data from port 2
```

13.9.13 **putexpio()** Output 1 Bit Data to Multiple Extended Multipurpose PIO Ports

[Description]
Outputs 1 bit data to multiple extended multipurpose PIO ports.

[Syntax]
putexpio(pioBits,outBits) ;

[Parameters]

pioBits	Bit string indicating the extended PIO port number (0x00 - 0xFF) Higher-order bit (Port number 7) <--- Lower-order bit (Port number 0) **Note**: Specify the target ports.
outBits	The output bit string corresponding to extended PIO port number bit sting Data (0x00 - 0xFF) **Note**: Any output against ports 4 – 7 is invalid.

[Return Value]
No return value

[Example]

```
putexpio(0xC5,0x81) ;          // output data to ports 2 and 0
```

13.9.14 **getexpio()** Get 1 Bit Data from Multiple Extended Multipurpose PIO Ports

[Description]
Gets 1 bit data of the corresponding port from extended multipurpose PIO ports.

[Syntax]
inBits = getexpio(pioBits) ;

[Parameters]

pioBits	Bit string indicating the extended PIO port number (0x00 - 0xFF) Higher-order bit (Port number 7) <--- Lower-order bit (Port number 0) **Note**: Specify the target ports.

[Return Value]

inBits	Success: returns retrieved bit corresponding data Failure: returns a value less than 0 **Note**: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

```
inBits = getexpio(0x05) ;          // get data from ports 2 and 0
```

13.9.15 **setexport()** Set I/O Direction of Extended Multipurpose PIO Port

[Description]
Sets the I/O direction of the specified extended multipurpose PIO port.

[Syntax]
rtn = setexport(portNo,dir) ;

[Parameters]

portNo	The extended PIO port number (0 - 7)
dir	The I/O direction (0: Input, 1: Output) **Note**: Any output against ports 4 - 7 is invalid.

[Return Value]

rtn	Success: returns 0 Failure: returns a value less than 0 **Note**: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

```
rtn = setexport(1,1) ;          // set the direction of port 1 to output
```

13.9.16 **setexpio()** Set I/O Direction of Extended Multipurpose PIO Ports

[Description]
Sets the I/O direction of extended multipurpose PIO ports.

[Syntax]
setexpio(pioBits,dirBits) ;

[Parameters]

pioBits	Bit string indicating the extended PIO port number (0x00 - 0xFF) Higher-order bit (Port number 7) <--- Lower-order bit (Port number 0) **Note**: Specify the target ports.
dirBits	Bit string indicating the I/O direction (0x00 - 0xFF) The I/O direction (0: Input, 1: Output) **Note**: Specifying any bits corresponding to ports 4 – 7 as the target is invalid.

[Return Value]

rtn	Success: returns 0 Failure: returns a value less than **Note**: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

```
rtn = setexpio(0xC5,0x81) ;          // set I/O direction of ports 7, 6, 2, and 0
```

13.9.17 **statexport()** Get I/O Direction of Extended Multipurpose PIO Port

[Description]
Gets the I/O direction of an extended multipurpose PIO port.

[Syntax]
dir = statexport(portNo) ;

[Parameters]

portNo	The number of an extended PIO port (0 - 7)

[Return Value]

dir	Success: returns the I/O direction (0: Input, 1: Output) Failure: returns a value less than **Note**: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

```
dir = statexport (0) ;
```

13.9.18 **statexpio()** Get I/O Direction of Extended Multipurpose PIO Ports

[Description]
Gets the I/O direction of extended multipurpose PIO ports.

[Syntax]
dirBits = statexpio(pioBits) ;

[Parameters]

pioBits	The number of extended PIO ports (0 - 7) Bit string indicating the port number (0x00 - 0xFF) Higher-order bit (Port number 7) <--- Lower-order bit (Port number 0) **Note**: Specify the target ports.

[Return Value]

dirBits	Success: returns a bit string indicating the I/O direction (0x00 - 0xFF) The I/O direction (0: Input, 1: Output) Failure: returns -1

[Example]

```
dirBits = statexpio (0xFF) ;          // get the I/O direction for ports 0 - 7
```

13.10 AD Conversion Functions

13.10.1 **getadc()** Get A/D Conversion Value

[Description]
Gets an A/D conversion value.

[Syntax]
inDat = getadc(chan) ;

[Parameters]

chan	A channel number (0 - 3)

[Return Values]

inDat	Success: returns a value of 0 or more (indicating the A/D conversion value) Failure: returns a value less than 0 **Note**: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

```
inDat = getadc(0) ;          // get the conversion value of channel 0
```

13.11 IIC I/O Functions

13.11.1 i2cput() Write Data into IIC

[Description]
Writes data into IIC.

[Syntax]
rtn = i2cput(devCode,addr,outDat,speed) ;

[Parameters]

devCode	A device code (0x00 - 0xFE) **Note**: Bit 0 is invalid.
addr	An address A device-specific address (0 - 255) **Note**: Refer to the connecting device specification before setting its address
outDat	Output data (0x00 - 0xFF)
speed	Speed The transfer frequency (100 KHz or 400 KHz) 100/400

[Return Value]

rtn	Success: returns 0 Failure: returns a value less than 0 **Note**: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

```
rtn = i2cput(0xA0,100,0x55,400) ;          // output 0x55 to the address of 100
```

Note: This function is provided for compatibility purposes only.
It is recommended to use **i2cwrite()** instead of using this function.

13.11.2 **i2cget()** Read Data from IIC

[Description]
Reads data from IIC.

[Syntax]
inDat = i2cget(devCode,addr,speed) ;

[Parameters]

devCode	A device code (0x00 - 0xFE) **Note**: Bit 0 is invalid.
addr	An address A device-specific address (0 - 255) **Note**: Refer to the connecting device specification before setting its address.
speed	Speed The transfer frequency (100 KHz or 400 KHz) 100/400

[Return Value]

inDat	Success: returns retrieved data (0x00 - 0xFF) Failure: returns a value less than 0 **Note**: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

```
inDat = i2cget(0xA0,100,400) ;          // read data from the address of 100
```

Note: This function is provided for compatibility purposes only.

It is recommended to use the **i2cread()** function instead of using this function.

13.11.3 **i2cset()** Set IIC Device Information

[Description]
Sets IIC device information.

[Syntax]
rtn = i2cset(devCode,adrB,speed,writeB,readB,[tmout]);

[Parameters]

devCode	A device code (0x00 - 0xFE) **Note**: Bit 0 is invalid.
adrB	The number of bytes for the address (1 or 2)
speed	The transfer frequency (100 KHz or 400 KHz) 100/400
writeB	The number of bytes that can be written in a row A value smaller than 0: the number of writable bytes in a row
readB	The number of bytes that can be read in a row 0: No limit A value smaller than 0: the number of readable bytes in a row
tmout	Timeout that applies when a sequential read/write operation is in process (specify a value of 10 or more in msec) **Note**: The value must be specified on a 10 msec basis because the first digit will be ignored.

The Default Values of Parameters

adrB	speed	writeB	readB	tmout
1	100	8	0	100

[Return Value]

rtn	Success: returns 0 Failure: returns a value less than 0 **Note**: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

```
rtn = i2cset(0x0A,2,400,64,0,100) ;
                                    // device code: 0x0A
                                    // number of bytes for the address: 2 bytes
                                    // transfer speed: 400KHz
                                    // number of writable bytes in a row: 64 bytes
                                    // number of readable bytes in a row: 0 (no limit)
                                    // timeout for read/write operation
                                    // in a row: 100 msec
```

13.11.4 i2cwrite() Output Multi-Byte of Contiguous Data to IIC

[Description]
Outputs multiple bytes of contiguous data to IIC.

[Syntax]
rtn = i2cwrite(devCode,addr,wBuf[],wCnt);

[Parameters]

devCode	A device code (0x00 - 0xFE) **Note**: Bit 0 is invalid.
addr	The write-to address The device-specific address (0 - 0xFFFF) **Note**: Refer to the connecting device specification before setting its address.
wBuf[]	An associative array containing the output data
wCnt	The number of output bytes

[Return Value]

rtn	Success: returns a value of 0 or more (indicating the number of output bytes) Failure: returns a value less than 0 **Note**: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

Writing Data to the Specified Address

```
rtn = i2cwrite(0xA2,1000,wBuf[],100) ;
// write 100 bytes of contiguous data at address 1,000 of the device code 0xA2
```

13.11.5 i2cread() Read Multi-Byte of Contiguous Data from IIC

[Description]
Reads multiple bytes of contiguous data from IIC.

[Syntax]
rtn = i2cread(devCode,addr,&rBuf[],rCnt);

[Parameters]

devCode	A device code (0x00 - 0xFE) **Note**: Bit 0 is invalid.
addr	A value of 0 or more: specify the read-from address The device-specific address (0x00 - 0xFFFF) **Note**: Refer to the connecting device specification before setting its address. -1: specify to read data from the current address inside the device
rBuf[]	An associative array for storing the retrieved data
rCnt	The number of bytes to be retrieved

[Return Value]

rtn	Success: returns a value of 0 or more (indicating the number of bytes read) Failure: returns a value less than 0 **Note**: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

1. Reading Data from the Specified Address

```
rtn = i2cread(0xA2,1000,&rBuf[],100) ;
// read 100 bytes of contiguous data from address 1,000 of the device code 0xA2
```

2. Reading Data from the Current Address

```
rtn = i2cerite(0xA2,-1,&rBuf[],100);
// read 100 bytes of contiguous data from the current
// address of the device code 0xA2
```

13.12 FTP Send/Receive Functions

13.12.1 **ftpput()** Send File via FTP

[Description]

Sends a file via FTP.

Note: The submit process of FTP starts when the script process is completed or paused.

For more information, see “Notes on the Use of Script” on page 221.

[Syntax]

```
rtn = ftpput(hostaddr,userid,passwd,remote_file,local_file,type) ;
```

[Parameters]

hostaddr	The host address
userid	User ID
passwd	Password
remote_file	The name of a remote file
local_file	The name of the local file
type	Data type (“A”/”a”: ASCII, “B”/”b”: BINARY)

[Return Value]

rtn	Success: returns 0 Busy :1 — the file is already under the process of sending via FTP and this request is not acceptable Failure: returns a value less than 0 **Note**: For more information about error codes, see “Detailed Information about Error Codes” on page 234

[Example]

1. Specifying the IP Address

```
rtn = ftpput(“xxx.xx.xx.xxx”,”wsc-user”,”wsc-passwd”,”test.img”,
“B:\\test.img”,”B”) ;
```

2. Specifying the Host Name (provided DNS is enabled)

```
rtn = ftpput(“prodigio”,”wsc-user”,”wsc-passwd”,”test.img”,
“B:\\test.img”,”B”) ;
```

Note: Connecting in PASV mode is not supported.

13.12.2 **ftpget()** Receive File via FTP

[Description]

Receives a file via FTP.

Note: The submit process of FTP starts when the script process is completed or paused.

For more information, see “Notes on the Use of Script” on page 221.

[Syntax]

rtn = ftpget(hostaddr,userid,passwd,remote_file,local_file,type) ;

[Parameters]

hostaddr	The host address
userid	User ID
passwd	Password
remote_file	The name of a remote file
local_file	The name of the local file
type	Data type (“A”/”a”: ASCII, “B”/”b”: BINARY)

[Return Value]

rtn	Success: returns 0 Busy : 1 — the file is already under the process of sending via FTP and this request is not acceptable Failure: returns a value less than 0 **Note**: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

1. Specifying the IP Address

```
rtn = ftpget(“xxx.xx.xx.xxx”,”wsc-user”,”wsc-passwd”,”test.img”,
“B:\\test.img”,”B”) ;
```

2. Specifying the Host Name (provided DNS is enabled)

```
rtn =ftpget(“prodigio”,”wsc-user”,”wsc-passwd”,”test.img”,
“B:\\test.img”,”B”) ;
```

Note: Connecting in PASV mode is not supported.

13.12.3 **stat_ftp()** Get Status of FTP Send/Receive

[Description]
Gets the sending and receiving status of FTP.

[Syntax]
rtn = stat_ftp();

[Parameters]
No parameters

[Return Value]
rtn Success: returns one of the following values indicating the status of FTP send/receive
1: FTP GET in progress
2: FTP PUT in progress
3: Initialized
a value less than 0: error detected
Note: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

```
rtn = statftp();          // rtn: the status of FTP send/receive
```

13.13. Send E-Mail Functions

13.13.1 sendmail() Send E-Mail

[Description]

Sends an E-mail.

Note: The E-mail submit process starts when the script process is completed or paused.

See details on “14. Notes on the Use of Script.”

[Syntax]

```
rtn = sendmail(fromAdr,toAdr,subjectPtr,bodyPtr,attachPtr) ;
```

[Parameters]

fromAdr	sender’s E-mail address (up to 128 characters)
toAdr	recipient’s E-mail address (up to 128 characters)
subjectPtr	Subject (up to 64 characters)
bodyPtr	Body text (up to 1,024 characters)
attachiPtr	The name of attachment file (optional) **Note**: Attachment file size is limited to 1 MB.

[Return Value]

rtn	Success: returns 0 Failure: returns a value less than 0 **Note**: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Examples]

1. A File Attached

```
rtn = sendmail(“info@prodigio.jp”,”xxxx@yyyy.zz.jp”,
“subject”,”body”,”B:\attach.txt”) ;
```

2. No File Attached

```
rtn = sendmail(“info@prodigio.jp”,”xxxx@yyyy.zz.jp”,
“subject”,”body”,””) ;
```

[Notes]

Depending on the SMTP server you are using to send mail, it may be necessary to make settings for POP before SMTP.
Please see [14.4 List of Network Parameters] for details about making these and other settings. If an error occurs while sending mail, please check the error code displayed in the debug monitor. For details about these error codes, see [14.10 Detailed Information about Error Codes] for more details.

13.13.2 **stat_smtp()**

[Description]
Retrieves the status of E-mail delivery.

[Syntax]
rtn=stat_

smtp();

[Parameters]
No parameters

[Return Value]
rtn Success: returns one of the following values indicating the status of message delivery
0: sent successfully
1: pending
2: body text contains something inappropriate
3: incorrect address
4: SMTP server not found
5: TCP error detected
6: canceled (reserved for future use)
7: communication error
-1: initial status

[Example]

```
rtn=stat_smtp();                    // rtn: the status of message delivery
```

13.14. Web Output Function

13.14.1 webprint() Output to Web

[Description]
Outputs data to web.

[Syntax]
rtn=webprint(webStr) ;

[Parameters]

webStr	The output string

[Return Value]

rtn	Success: returns 0 Failure: returns -1

[Example]

```
rtn = webprint("<html><body>Test</body></html>") ;
    // output "test" to the browser
```

13.15. Event Process Function

13.15.1 **event()** Register Events

[Description]

Registers timer events, RTC events, SIO events, and PIO events.

[Syntax]

```
rtn = event(id,scrf,prm,trg1,trg2,exec_no,wdt,[pri]);
```

[Parameters]

id — Event Type

“TIMER”	: Timer Events
“RTC”	: RTC Events
“SIO”	: SIO Events
“PIO”	: PIO Events
“PULSE”	: PIO Pulse Events
“USER”	: User Software Events

scrf — The name of a script file (must be specified in fully qualified path)

prm — A parameter string
specify parameters to be passed on to the script (use spaces as delimiters.)

Example: “parameter_name1=param1 parameter_name2=param2 ...”
//parameter_name1...: name of parameter
//param1...: value of parameter

trg1 — Trigger 1

- TIMER	the value of a trigger timer (100 - , in msec)	The value of a trigger timer value needs to be specified on a 100 msec basis because its value is rounded down to hundredths.
- RTC	: time string (Mandatory)	The string is to be specified in accordance with the following format: “hh:mm:ss.” Hours (HH) are to be specified in 24-hour format. hh and mm are optional. “hh:mm:ss”: boots at the specified time (second, minute, hour) “mm:ss”: boots at the specified minute of the hour “ss”: boots at the specified second of the minute
- SIO	: A channel number (1 - 2)	
- PIO	: A port number (0 - 15)	
- PULSE	: A port number (0 - 15)	
- USER	: Fixed at 0	

trg2 — Trigger 2

- TIMER	: Fixed at 0

	- RTC	: time string (Optional. Use “” when omitting)	The string is to be specified in accordance with the following format: “YY/MM/DD” Year (YY) is to be specified in two-digit year. YY and MM are optional. “YY/MM/DD”: boots on the specified date (day, month, year) “MM/DD”: boots on the specified date of the year (in the form of MM, DD) “DD”: boots on the specified date of the month “”: boots daily **Note**: This works in conjunction with the time criteria specified with trg1.
	- SIO	: the number of bytes to receive	
	- PIO	0: Low level [changes from High to Low] 1: High level [changes from Low to High]	
	- PULSE	: A number equal to or larger than 100: pulse monitoring time (in msec)	The pulse monitoring time is to be specified on a 100 ms basis because its value is rounded down to hundredths.
	- USER	: Fixed at 0	
exec_no	- The number of executions	0: will not be executed -1: will be executed infinitely a value less than 0: will be executed for the specified number of times	In “USER” event, this has a fixed value of 1.
wdt	- monitoring time (in msec) Specify the time for monitoring script. Terminates the script if the process does not finish within the specified period of time. The value of an event monitoring time needs to be specified on a 1,000 msec basis because its value is rounded down to thousandths.		
pri	- Priority of execution	1 (High) - 5 (Low)	Optional: When omitted, priority 5 is assumed.

[Return Value]

rtn Success: returns 0
Failure: returns a value less than 0

Note: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Examples]

1. Timer Event
```
event(“TIMER”,”C:\\script\\test.wcr”,”ABC=1 DEF=2”,1000,0,1,10000);
// execute once a second
```

2. RTC Event
```
event(“RTC”,”C:\\script\\test.wcr”,”ABC=1 DEF=2”,”10:31:25”,”03/12/24”,1, 10000);
// execute once at the specified time
```

3. SIO Event
```
event(“SIO”,”C:\\script\\test.wcr”,”ABC=1 DEF=2”,1,16,1, 10000) ;
// get 16 bytes of data once from channel 1
```

4. PIO Event
```
event(“PIO”,”C:\\script\\test.wcr”,”ABC=1 DEF=2”,1,0,1,10000);
// output zero once to port 1
```

5. PULSE Event
```
event(“PULSE”,”C:\\script\\test.wcr”,”ABC=1 DEF=2”,1,200,1, 10000) ;
// monitor pulse for port 1 at 200 msec intervals and execute the process once
```

6. USER Event
```
event(“USER”,”C:\\script\\test.wcr”,”ABC=1 DEF=2”,0,0,1, 10000) ;
// register the script file containing arguments to an event queue
```

Note: ProDigio will be reset if a monitoring timeout occurs while the event function is running.
Please be cautious not to enter an infinite loop on a process error.

13.16 System Control Functions

13.16.1 reboot() Reboot System

[Description]
Reboots the system after waiting for the specified period of time.

[Syntax]
rtn = reboot(waitTime) ;

[Parameters]

waitTime	Wait time (in msec)

[Return Value]

rtn	Success: returns 0 Failure: returns a value less than 0

Note: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

```
rtn = reboot(1000) ;        // reboot in a second
```

13.16.2 **getwscparam()** Get Parameter

[Description]
Gets a startup (boot) parameter.

[Syntax]
valstr = getwscparam(flg,prmstr) ;

[Parameters]

flg	The flag for the parameter to get. Specify one of the following values as appropriate: 0: Factory default value 1: Startup parameter setting value 2: Current value
prmstr	ID string of the parameter to get

[Return Value]

valstr	Success: returns the retrieved parameter string Failure: returns -1 or NULL (no value set)

[Example]

1. Directly Specifying the String

```
varstr = getwscparam(2, “sntp_enable”) ;      // get SNTP status strings from
                                              // the retrieved parameter information
```

2. Specifying the Variable

```
setstr = “sntp_enable” ;
flg = 2 ;
varstr = getwscparam(flg, setstr) ;           // varstr: the current value of “sntp_enable”
```

[Remarks]
For more information about parameters, see the list of parameters in “14. Appendix.”

13.16.3 **setwscparam()** Set Parameter

[Description]
Sets a startup (boot) parameter.

[Syntax]
rtn = setwscparam(flg,prmstr,valstr) ;

[Parameters]

flg	Flag for the setting parameter. Specify one of the following values as appropriate: 0: Factory default value 1: Startup parameter setting value
prmstr	ID string of the parameter to get
valstr	The parameter string to be set

[Return Value]

rtn	Success: returns 0 Failure: returns a value less than 0 (the setting value is incorrect or invalid) **Note**: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

1. Directly Specifying the String

```
rtn = setwscparam(1, “sntp_enable”, “off”) ;
        // set the startup parameter “sntp_enable” to “off”
```

2. Specifying the Variable

```
flg = 1 ;
prmstr = “sntp_enable” ;
valstr = “off” ;
rtn = setwscparam(flg, prmstr, valstr) ;
// set the startup parameter “sntp_enable” to “off”
```

[Remarks]
For more information about parameters, see the list of parameters in “14. Appendix.”

13.16.4 **storewscparam()** Store Parameter in Flash ROM

[Description]
Stores a parameter in flash ROM.

[Syntax]
rtn = storewscparam(flg) ;

[Parameters]

flg	Flag for the setting parameter. Specify one of the following values as appropriate: 0: Factory default value 1: Startup parameter setting value

[Return Value]

rtn	Success: returns 0 Failure: returns a value less than 0 **Note**: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

```
rtn = storewscparam(1) ;          // store the startup parameter setting value
```

13.16.5 **crtwscparam()** Create Parameter File

[Description]
Creates a parameter file.

[Syntax]
rtn = crtwscparam([flg]) ;
Note: A value enclosed in [ ] is optional.

[Parameters]

flg	Flag for the setting parameter. Specify one of the following values as appropriate: 0: Factory default value 1: Startup (boot) parameter setting value 2: Current value

[Return Value]

rtn	Success: returns 0 Failure: returns a value less than 0 **Note**: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

1. No Parameter Specified (Startup (Boot) Parameter Setting Value)
   rtn = crtwscparam() ; // create a parameter file from the current value
2. Factory Default Value
   rtn = crtwscparam(0): // create a parameter file from the factory default value
3. Startup (Boot) Parameter Setting Value
   rtn = crtwscparam(1): // create a parameter file from the startup (boot)
   // parameter setting value

[Remarks]

The created parameter file will be C:\system\wscparam.sys file.
For more information about parameters, see the list of parameters in “14. Appendix.”

13.16.6 **reiniwscparam()** Reset to Factory Default Value

[Description]
Resets parameters to the factory default values.

[Syntax]
rtn = reiniwscparam() ;

[Parameters]
No parameters

[Return Value]

rtn	Success: returns 0 Failure: returns a value less than 0

Note: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

```
rtn = reiniwscparam() ;          // reset parameters to the factory default values
```

13.16.7 **getmacaddr()** Get MAC Address

[Description]
Gets a Mac address.

[Syntax]
rtn = getmacaddr(&mac[ ]) ;

[Parameters]
mac[ ] An associative array for storing a MAC address

[Return Value]

str	returns a MAC address string 0: failed to get the MAC address
rtn	Success: returns 0 Failure: returns a value less than 0

Note: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

```
var mac[ ] ;
rtn = getmacaddr(&mac[ ]) ;        // get a MAC address
```

[Remarks]

An Example of Displaying a MAC Address

```
var mac[];
rtn = getmacaddr(&mac[]);
puts(“\n%hexstr(mac[0])%:%hexstr(mac[1])%:%hexstr(mac[2])%:
        %hexstr(mac[3])%:%hexstr(mac[4])%:%hexstr(mac[5])%”);
```

13.16.8 **phpowctrl()** Control Power

[Description]
Controls the power supply (ON or OFF) of USB-HOST and PCMCIA.

[Syntax]
rtn = phpowctrl (dev,ctrl) ;

[Parameters]

dev	The type of device. Specify one of the following values as appropriate: 1: USB-HOST 2: PCMCIA (Slot 0) 3: PCMCIA (Slot 1)
ctrl	Control Information about power supply. Specify one of the following values as appropriate: 0: turn the power supply off 1: turn the power supply on

[Return Value]

rtn	Success: returns 0 Failure: returns -1

[Example]

1. USB-HOST

```
rtn = phpowctrl(1,1) ;          // turn the USB-HOST power supply on
```

2. PCMCIA

```
rtn = phpowctrl(2,0) ;          // turn the PCMCIA (Slot 0) power supply off
```

[Remarks]
It takes time for the script to be aware of the change in the status of a device. The time varies according to the hardware in use. (The ProDigio Training Board takes about 2 to 5 seconds to recognize the change.)

13.16.9 **get_module_sw()** Get DIP-SW on Module

[Description]
Gets a DIP-SW value on a module.

[Syntax]
swval = get_module_sw() ;

[Parameters]
No parameters

[Return Value]
swval Success: returns a DIP-SW value (4 bit: 0x00 - 0x0F)
Higher-SW (3) <--- Lower-SW (0)

[Example]

```
swval = get_module_sw() ;          // Get a DIP-SW value
```

13.16.10 **outeventlogmsg()** Output Message to Event Log

[Description]

Outputs a message to the event log.

Note: You can view messages logged to the event log by issuing the evtdump command on the debug monitor.

[Syntax]

```
rtn = outevtlogmsg(msgStr) ;
```

[Parameters]

msgStr — A message string to be output (maximum of 76 characters)

Note: If a message is longer than 76 characters, only the first 76 characters will be logged into the event log.

[Return Value]

rtn — Success: outputs a message to the event log
Failure: returns a value less than 0

Note: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Examples]

```
rtn = outevtlogmsg(“error_message”) ;          // output the string to the event log
```

13.16.11 **getsysver()** Get the system version information character string

[Description]
Get the system version information character string

[Syntax]
rtn = getsysver() ;

[Parameters]
None

[Return Value]

rtn	Success: Version Info Character String
	Failure: returns a NULL(“ “).

[Examples]

```
rtn = getsysver() ;      // Get the system version information character string “01.07.00“
```

13.17 Dialing Control Functions

13.17.1 start_atdialup() Start isp/smtp Line Connection

[Description]

Starts dial-up connection to the specified service provider by using the preset dial-up parameters. **Note**: You can specify only one connection, either isp or smtp.

<Dial-up Parameters>

You need to set the following connection parameters in advance:

1. isp - Internet Service Provider

	Parameter Name	Description
1	isp_dial_no	ISP dial number
2	isp_user_id	ISP user ID
3	isp_passwd	ISP password
4	isp_dns_addr	ISP DNS server address
5	isp_ppp_auth	ISP PPP communication authentication mode
6	isp_self_addr	ISP client IP address (*1)

2. smtp - Mail Service

	Parameter Name	Description
1	smtp_dial_no	SMTP dial number
2	smtp_user_id	SMTP user ID
3	smtp_passwd	SMTP password
4	isp_smtp_dns	SMTP DNS server address
5	smtp_ppp_auth	SMTP PPP Communication Authentication Mode
6	smtp_self_addr	SMTP client IP address (*1)

*1 These parameters are required when you need to specify the client IP address. In most cases, a NULL is specified to automatically retrieve the IP address.

[Syntax]

```
rtn = start_atdialup (sptype,incom) ;
```

[Parameters]

sptype	Type of the service provider	
	0: isp	Internet service
	1: smtp	Mail service
incom	Connection mode	
	0: Outgoing	
	1: Incoming	

Notes: Once you set the connection mode to incoming mode, it stays in effect until you reset the system or issue a disconnection function while connected.

If you want to use a dial-up connection in incoming mode, set the parameter for the incoming mode to ON.

[Return Value]

rtn	Success: returns 0 (connection established)
	Failure: returns a value less than 0

Note: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

1. Connecting to ISP

```
rtn = start_atdialup(0,0) ;              // connect to ISP
```

2. Connecting to ISP via SMTP

```
rtn = start_atdialup (1,0) ;             // connect via smtp
```

3. Waiting for Incoming Connection from ISP

```
rtn = start_atdialup(0,1) ;
// wait for incoming connection by using ISP dial-up parameters
```

4. Waiting for Incoming Connection via SMTP

```
rtn = start_atdialup (1,1) ;
// wait for the incoming connection by using SMTP dial-up parameters
```

13.17.2 stop_atdialup() Disconnect Communication Line

[Description]

Disconnects the communication line.

Note: In case of waiting for incoming connection, the incoming mode will be disabled.

[Syntax]

```
rtn = stop_atdialup () ;
```

[Parameters]

No parameters

[Return Value]

rtn	Success: returns 0 Failure: returns a value less than 0

Note: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

```
rtn = stop_atdialup () ;
```

13.17.3 **stat_atconnect()** Get Status

[Description]
Gets the status of a line.

[Syntax]
rtn = stat_atconnect () ;

[Parameters]
No parameters

[Return Value]
rtn Success: returns one of the following values indicating the status of a line
0: Not connected
1: Checking the modem
2: Running initialization AT command
3: Dialing
4: Waiting for incoming connection
5: Connecting
6: Communicating
7: Disconnecting
8: Disconnected
Failure: returns -1

[Example]

```
rtn = stat_atconnect () ;          // returns the current status of a line
```

13.17.4 **setini_atcommand()** Set Initialization AT Command

[Description]

Sets a customized initialization AT command string.

Notes:

The maximum number of initialization AT commands that can be registered is 8.

The specified command strings are performed in the order they are registered when attempting to establish a dial-up connection.

Any commands you set with this function stay in effect until you reset the system or execute this function.

If the number of commands is larger than seven, all of its settings will be cleared.

[Syntax]

```
rtn = setini_atcommand (seqno,atinistr) ;
```

[Parameters]

seqno	The order in which the commands are performed (0 - 7)
atinistr	The initialization AT command string (31 characters maximum)

[Return Value]

rtn	Success: returns 0 Failure: returns a value less than 0 **Note**: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

Specify command strings that you need. If you do not need eight commands, specify a NULL (“ “) for the second parameter.

```
rtn = setini_atcommand (0,”AT&F”) ;          // &F command
rtn = setini_atcommand (1,”AT&E0”) ;         // &En command
rtn = setini_atcommand (2,””) ;
                :
rtn = setini_atcommand (7,””) ;
```

13.17.5 setdial_atcommand() Set Dial-up AT Command

[Description]

Sets an AT command for establishing a dial-up connection.

Note: Specify the AT command (including parameters) that you want to specify prior to the phone number.

[Syntax]

rtn = setdial_atcommand (dailstr,optstr) ;

[Parameters]

dialstr	A Dial-up AT command string (31 characters maximum) **Note**: dialstr is optional. If not specified, this defaults to “ATD.”
optstr	The communication mode option (such as “#32”, “#64”) **Note**: optstr is optional

[Return Value]

rtn	Success: returns 0 Failure: returns a value less than 0 **Note**: For more information about error codes, see “Detailed Information about Error Codes” on page 234.

[Example]

1. dialstr and optstr Specified

```
rtn = setdial_atcommand (“ATDT”,”#32”) ;     // 32KB data communication mode
```

2. optstr Not Specified

```
rtn = setdial_atcommand (“ATDT”) ;           // dial-up using ATDT
```

3. dialstr and optstr Not Specified

```
rtn = setdial_atcommand () ;                 // dial-up using “ATD”
```

13.17.6 **get_atispno()** Get Line Type

[Description]
Gets the type of a line.

[Syntax]
rtn = get_atispno() ;

[Parameters]
No parameters

[Return Value]
rtn Success: returns one of the following line types
0: isp
1: smtp
-1: Initial state
Note: If you get a value of initial state, it means you have never used this communication line in the past.

[Examples]

```
rtn = get_atispno() ;
```

13.17.7 **get_atresult()** Get Result Code

[Description]
Gets a result code when an error occurred on a dial-up connection.

[Syntax]
rtn = get_atresult() ;

[Parameters]
No parameters

[Return Value]
str Success: returns an error result code string
<Standard Error Result Code Strings>
“NO CARRIER”
“ERROR”
“NO DIALTONE”
“BUSY”
“NO ANSWER”
“DELAYED”
“BLACKLISTED”

Note: If you have set custom error result code strings, one of those strings will be returned.

Failure: returns a NULL string

If there is no reply from the modem, the function returns a string of “ERROR.”

[Examples]

1. Connection Error
str = get_atresult() ; // str: “NO CARRIER”

13.17.8 **seterr_atresult()** Set Result Code String

[Description]

Sets custom error result code strings.

Notes:

The maximum number of error result code strings that can be registered is 8.

You can register the device-specific result codes other than the following standard error code strings:

<Standard Error Result Code Strings>

"NO CARRIER"

"ERROR"

"NO DIALTONE"

"BUSY"

"NO ANSWER"

"DELAYED"

"BLACKLISTED"

Note: Any commands you set with this function stay in effect until you reset the system or execute this function.

If the number of a commands is larger than seven, all of its settings will be cleared.

[Syntax]

```
rtn = seterr_atresult (seqno,aterrstr) ;
```

[Parameters]

seqno	The number specified for each result code string (0 - 7)
aterrstr	An error result code string specific to a device (up to 31 characters)

[Return Value]

rtn	Success: returns 0 Failure: returns -1

[Examples]

Specify error result code strings that you need. If you do not need eight codes, specify a NULL (" ") for those that do not contain any code strings.

```
rtn = seterr_atresult (0,"OUT OF SERVICE") ;          // out of service
rtn = seterr_atresult (1,"NOT CORRECT") ;             // data attribute is not correct
rtn = seterr_atresult (2,"") ;
            :
rtn = setini_atcommand (7,"") ;
```

14. Appendix

14.1 Notes on the Use of Script

The following table contains a list of points you need to pay attention to when using WSC script. Please read the provided information carefully as it contains important information such as file size limitations.

No.	Item	Restriction	Remarks
1	Script File Size	The maximum size of a script file that you can load at a time is 32 KB (including comments.) If a file is larger than 32 KB, the script loads only up to the first 32 KB of the file.	
2	Nested Control Structure	The control structure supports nesting up to the five levels deep. If it exceeds this limit, the script terminates with an error.	
3	Script Parameter	You can pass up to 4 KB of parameters, if script parameters are automatically generated by CGI.	
4	File I/O	If you perform file I/O using the functions listed in the Remarks column, the maximum I/O size at a time is 4,096 bytes. If the specified size exceeds the limit of 4,096 bytes, they input/output up to the first 4,096 bytes. If you need to perform file I/O larger than 4,096 bytes, call the appropriate function recursively until all data is processed.	**Relevant Functions** filereadhash() filewritehash() fileread() filewrite()
5	Flash Files	Turning on modules DSW1-1 and DSW3-5 will protect the flash file drive (“C:\”) from writing. You may lose your data stored in a flash file if you press the reset button or turn the power off while writing data into the flash file.	
6	Running Scripts from Browser	You cannot directly call and run a script file from the browser. If you are to use the script file via CGI, make sure that you call and run the file from an HTML file.	
7	Event Registration	Irrespective of event types, the maximum number of events that you can register is 64. Please remember that attempting to register more than 64 events will results in an error.	
8	Sending E-Mail	When sending E-mails in a row, the process of sending the first E-mail must be completed for the next subsequent E-mail to be sent. Before attempting to send the next E-mail, use stat_smtp() function to check that the first E-mail has been sent successfully. E-mails must be sent after script processing is completed or you need to use the combination of sleep function and stat_smtp function to adjust the timing of status monitoring.	
9	Serial Event	Registering different events on a serial port may cause unexpected behavior. Additionally, make sure that events registered on a serial port are not duplicated.	
10	FTP Send and Receive	When delivering data via FTP in a row, the process of first delivery must be completed before the next delivery takes place. If you call FTP Send and Receive Functions consecutively, the return value for subsequently called functions will be 1 (busy). Before conducting the next process of FTP delivery, use statftp(), Get Status of FTP Send/Receive function, to check that the preceding FTP delivery has been successfully completed. The FTP delivery process should be started after script processing is completed or while the script process is temporarily paused by the sleep function. If you are to call the Get Status of FTP Send/Receive, stat_ftp(), function to monitor the status of FTP delivery, use the combination of sleep function and stat_ftp() function to adjust the timing of status monitoring.	

14.2 List of Error Messages

The following table lists the error messages that are returned from the debug monitor when executing WSC script:

No.	Error Message	Description
1	Script not Found	Could not find the script.
2	Function not Found	Could not find the function.
3	Function error	Function error.
4	Argument error	An argument with no value was found.
5	Too few Argument	There are not enough arguments.
6	Argument format error	The first argument is not specified in regular expression.
7	Sentence error	Syntax error.
8	Variable is not defined	The variable is not defined.
9	Declaration overlaps	Duplicate declaration exists.
10	Index error	The index is invalid.
11	Parenthes error	An opening or closing parenthesis is missing.
12	Quart error	An opening or closing quotation mark (“) is missing.
13	‘if’ and ’else’ do not correspond	if-else statement correspondence error.
14	Divide by ZERO	Dividing by zero is an invalid operation.
15	Illegal operator	The operator is invalid.
16	Illegal operator to string	An illegal operator is used in the string.
17	Illegal operator to hash	An illegal operator is used in the associative array.
18	File open error	Could not open the file.
19	Alloc failed	Failed to allocate memory.
20	Invalid discriptor	The descriptor is invalid.

<Message Format>
<WCR>:Function error:[Script File Name].[Function Name]:[Function Name]:Argument error at [Script File Name]. [Function Name]:(line number in which an error occurred) : [Statement]

14.3 List of Startup Parameters

The following tables provide lists of startup parameter settings, that can be specified in reading operations, and their default values:

<Settings>

No.	Item	Parameter Name	Format and Setting Value	Area Length
1	Default gateway address	gateway_addr	xxx.xxx.xxx.xxx xxx: Numbers (000 - 255)	15
2	DHCP Status	dhcp_enable	on/off on: Enabled off: Disabled	15
3	Self IP Address	ip_addr_self	xxx.xxx.xxx.xxx xxx: Numbers (000 - 255)	15
4	Subnet Mask	ip_net_mask	xxx.xxx.xxx.xxx xxx: Numbers (000 - 255)	15
5	Server IP Address	ip_addr_serv	xxx.xxx.xxx.xxx xxx: Numbers (000 - 255)	15
6	Server User ID	user_id_load	User ID string	15
7	Server User Password	passwd_load	User password string	15
8	File Name of the Control Program	pgm_file_name	File name string	63

<Default Values>

No.	Item	Parameter Name	Initial value (Default)
1	Default Gateway Address	gateway_addr	(NULL)
2	DHCP Status	dhcp_enable	off
3	Self IP Address	ip_addr_self	192.168.1.200
4	Subnet Mask	ip_net_mask	255.255.0.0
5	Server IP Address	ip_addr_serv	192.168.1.100
6	Server User ID	user_id_load	wsc-user
7	Server User Password	passwd_load	wsc-passwd
8	File Name of the Control Program	pgm_file_name	wsceva.bin

14.4 List of Network Parameters

<Settings>

No.	Item	Parameter Name	Format and Setting Value	Area Length
1	DNS Status	dns_enable	on/off on: Enabled off: Disabled	15
2	DNS Server IP Address (Primary)	dns_server1_addr	xxx.xxx.xxx.xxx xxx: Numbers (000 - 255)	15
3	DNS Server IP Address (Secondary)	dns_server2_addr	xxx.xxx.xxx.xxx xxx: Numbers (000 - 255)	15
4	SNTP Status	sntp_enable	on/off on: Enabled off: Disabled	15
5	SNTP Server IP Address (Primary)	sntp_server1_addr	xxx.xxx.xxx.xxx xxx: Numbers (000 - 255)	15
6	SNTP Server IP Address (Secondary)	sntp_server2_addr	xxx.xxx.xxx.xxx xxx: Numbers (000 - 255)	15
7	IP Block Feature Status	ip_block_enable	on/off on: Enabled off: Disabled	15
8	IP Block: Client IP address_1	ipb_client1_enable	xxx.xxx.xxx.xxx xxx: Numbers (000 - 255)	15
9	IP Block: Client IP address_2	ipb_client2_enable	xxx.xxx.xxx.xxx xxx: Numbers (000 - 255)	15
10	IP Block: Client IP address_3	ipb_client3_enable	xxx.xxx.xxx.xxx xxx: Numbers (000 - 255)	15
11	WSC Login ID	login_id	Login ID string	15
12	WSC Login Password	login_passwd	Login password string	15
13	Setting Tool Login ID	web_login_id	Tool login ID string	15
14	Setting Tool Password	web_passwd	Tool login password string	15
15	SMTP Server Name (Domain/IP Address)	smtp_server_name	Domain name or IP address Domain name: String IP address: xxx.xxx.xxx.xxx Numbers (000 - 255)	63
16	POP before SMTP Status	pop3_enable	on/off on: Enabled off: Disabled	15
17	POP Server Name (Domain/IP Address)	pop3_server_name	Domain name or IP address Domain name: String IP address: xxx.xxx.xxx.xxx Numbers (000 - 255)	63
18	POP Account name(User ID)	pop3_user_id	Account name string	47
19	POP Login password	pop3_passwd	Login password string	31
20	SMTP Server Communication Route (LAN/DIAL)	smtp_route	LAN/DIAL LAN: Wired LAN DIAL: PPP communication	15
21	Onboard LAN Interface	ether_speed	10, 100, auto 10 : 10 Mbps 100 : 100 Mbps auto: 10 or 100 Mbps auto-detection	15
22	Wireless LAN Status	wlan_enable	on or off on: Enabled off: Disabled	15
23	Wireless LAN Group ID	wlan_group_id	String	32
24	Wireless LAN WEP Key Type	wlan_wep_hex	on or off on: hexadecimal off: string	
25	Wireless LAN WEP Key Code	wlan_wep_key	String or hexadecimal string String specified per WEP key type: 5 or 13 characters Hexadecimal string: 10 or 26 characters	31
26	TCP-Serial Status	tcp_serial_enable	on or off on: Enabled off: Disabled	15

27	TCP-Serial COM Port	tcp_com_port	com1 or com2 com1: COM1 port com2: COM2 port	15
28	TCP-Serial COM Incoming Data Monitoring Time (in msec)	tcp_serial_time	Numerical (in msec) 1 - 65,535	15
29	TCP-Serial Port Number	tcp_serial_port	Numerical 1 - 65,535	10
30	ftp Port Number	ftp_port	Numerical 1 - 65,535	10
31	DTP (FTP) Port Number	dtp_port	Numerical 1 - 65,535	10
32	Server-side FTP Port	svr_ftp_port	Numerical 1 - 65,535	10
33	Server-side DTP Port	svr_dtp_port	Numerical 1 - 65,535	10
34	SMTP Port Number	smtp_port	Numerical 1 - 65,535	10
35	POP Port Number	pop3_port	Numerical 1 - 65,535	10
36	SNTP Port Number	sntp_port	Numerical 1 - 65,535	10
37	HTTP Port Number	http_port	Numerical 1 - 65,535	10
38	Script Communication Port Number	script_port	Numerical 1 - 65,535	10
39	Image Transfer Port Number Note: Currently not in use.	image_port	Numerical 1 - 65,535	10
40	Diag Port Number	diag_port	Numerical 1 - 65,535	10
41	UDP Setup - Server Port Number	cmd_server_port	Numerical 1 - 65,535	10
42	UDP Setup - Self Port Number	cmd_my_port	Numerical 1 - 65,535	10

<Default Values>

No.	Item	Parameter Name	Initial Value (Default)
1	DNS Status	dns_enable	off
2	DNS Server IP Address (Primary)	dns_server1_addr	(NULL)
3	DNS Server IP Address (Secondary)	dns_server2_addr	(NULL)
4	SNTP Status	sntp_enable	off
5	SNTP Server IP Address (Primary)	sntp_server1_addr	(NULL)
6	SNTP Server IP Address (Secondary)	sntp_server2_addr	(NULL)
7	IP Block Feature Status	ip_block_enable	off
8	IP Block: Client IP address_1	ipb_client1_enable	(NULL)
9	IP Block: Client IP address_2	ipb_client2_enable	(NULL)
10	IP Block: Client IP address_3	ipb_client3_enable	(NULL)
11	WSC Login ID	login_id	wsc-user
12	WSC Login Password	login_passwd	wsc-passwd
13	Setting Tool Login ID	web_login_id	"web-user"
14	Setting Tool Password	web_passwd	"web-passwd"
15	SMTP Server Name (Domain/IP Address)	smtp_server_name	(NULL)
16	POP before SMTP Status	pop3_enable	off
17	POP Server Name (Domain/IP Address)	pop3_server_name	(NULL)
18	POP Account name(User ID)	pop3_user_id	(NULL)
19	POP Login Password		
20	SMTP Server Communication Route (LAN/DIAL)	smtp_route	lan
21	Onboard LAN Interface	ether_speed	10
22	Wireless LAN Status	wlan_enable	OFF
23	Wireless LAN Group ID	wlan_group_id	(NULL)
24	Wireless LAN WEP Key Type	wlan_wep_hex	OFF
25	Wireless LAN WEP Key Code	wlan_wep_key	(NULL)
26	TCP-Serial Status	tcp_serial_enable	OFF
27	TCP-Serial COM Port	tcp_com_port	(NULL)
28	TCP-Serial COM Incoming Data Monitoring Time (in msec)	tcp_serial_time	300
29	TCP-Serial Port Number	tcp_serial_port	8100
30	ftp Port Number	ftp_port	21
31	DTP (FTP) Port Number	dtp_port	20
32	Server-side FTP Port	svr_ftp_port	21
33	Server-side DTP Port	svr_dtp_port	20
34	SMTP Port Number	smtp_port	25
35	POP Port Number	pop3_port	110
36	SNTP Port Number	sntp_port	123
37	HTTP Port Number	http_port	80
38	Script Communication Port Number	script_port	8101
39	Image Transfer Port Number	image_port	8003
40	Diag Port Number	diag_port	8002
41	UDP Setup - Server Port Number	cmd_server_port	62015
42	UDP Setup - Self Port Number	cmd_my_port	62017

14.5 List of Serial Parameters

<Settings>

No.	Item		Parameter Name	Format and Setting Value	Area Length
1	Communication Speed	COM0	com0_speed	600/1200/2400/4800/9600/19200/38400/115200 600 : 600bps 1200 : 1,200bps 2400 : 2,400bps 4800 : 4,800bps 9600 : 9,600bps 19200 : 19,200bps 38400 : 38,400bps 115200 : 115,200bps	15
2	Data Bit Length		com0_data_lan	7 or 8 7: 7 bits 8: 8 bits	15
3	Parity Bit		com0_parity_bit	none, even, or odd none: No parity even: Even parity odd: Odd parity	15
4	Stop Bit		com0_stop_bit	1 or 2 1: 1 bit 2: 2 bits	15
5	Communication Speed	COM1	com1_speed	600/1200/2400/4800/9600/19200/38400/115200 600 : 600bps 1200 : 1,200bps 2400 : 2,400bps 4800 : 4,800bps 9600 : 9,600bps 19200 : 19,200bps 38400 : 38,400bps 115200 : 115,200bps	15
6	Data Bit Length		com1_data_lan	7 or 8 7 : 7 bits 8 : 8 bits	15
7	Parity Bit		com1_parity_bit	none, even, or odd none: No parity even: Even parity odd: Odd parity	15
8	Stop Bit		com1_stop_bit	1 or 2 1: 1 bit 2: 2 bits	15
9	H/W Flow Control		com1_flow_ctl	on or off on: H/W flow control enabled off: H/W flow control disabled	15

10	Communication Speed	COM2	com2_speed	600/1200/2400/4800/9600/19200/38400/115200 600 : 600bps 1200 : 1,200bps 2400 : 2,400bps 4800 : 4,800bps 9600 : 9,600bps 19200 : 19,200bps 38400 : 38,400bps 115200 : 115,200bps	15
11	Data Bit Length		com2_data_lan	7 or 8 7 : 7 bits 8 : 8 bits	15
12	Parity Bit		com2_parity_bit	none, even, or odd none: No parity even: Even parity odd: Odd parity	15
13	Stop Bit		com2_stop_bit	1 or 2 1: 1 bit 2: 2 bits	15
14	H/W Flow Control		com2_flow_ctl	on or off on: H/W flow control enabled off: H/W flow control disabled	15
15	ER, DR Control		com2_sigline_ctl	on or off on: Enabled off: Disabled	15

<Default Values>

No.	Item		Parameter Name	Initial value (Default)
1	Communication Speed	COM0	com0_speed	9600
2	Data Bit Length		com0_data_lan	8
3	Parity Bit		com0_parity_bit	none
4	Stop Bit		com0_stop_bit	1
5	Communication Speed	COM1	com1_speed	9600
6	Data Bit Length		com1_data_lan	8
7	Parity Bit		com1_parity_bit	none
8	Stop Bit		com1_stop_bit	1
9	H/W Flow Control		com1_flow_ctl	off
10	Communication Speed	COM2	com2_speed	9600
11	Data Bit Length		com2_data_lan	8
12	Parity Bit		com2_parity_bit	none
13	Stop Bit		com2_stop_bit	1
14	H/W Flow Control		com2_flow_ctl	off
15	ER, DR Control		com2_sigline_ctl	off

14.6 List of Other Device Parameters

<Settings>

No.	Item	Parameter Name	Format and Setting Value	Area Length
1	I2C Status	i2c_enable	on or off on: Enabled off: Disabled	15
2	ADC Status	adc_enable	on or off on: Enabled off: Disabled	15
3	USB/IF Drive	usb_drv	C:\, D:\, or E:\ C:\ - Flash drive D:\ - CF-ATA E:\ - USB-ATA	15
4	USB Host Status Flag	usb_host	on or off on: Enabled off: Disabled	15
5	Serial Host I/F Status	siohost_enable	on or off on: Enabled off: Disabled	15
6	Serial Host I/F COM Port	siohost_com	COM1 or COM2 COM1: COM1 port COM2: COM2 port	15
7	GPIO I/O Setting	gpio_select	nnnnnnnnnnnnnnnn 15 . . . . . 0 n: Port setting (0 or 1) 0: Input 1: Output	16
8	GPIO Output Drive Capability	gpio_drv	pppppppppppppppp 15 . . . . . 0 p: Port setting (0 or 1) 0: Low (2mA) 1: High (10mA)	16
9	GPIO Signal Level	gpio_data	ssssssssssssssss 15 . . . . . 0 s: Signal level (0 or 1)	16
10	EXIO I/O Settings	exio_select	nnnnnnnnnnnnnnnn 15 . . . . . 0 n: 0 or 1 0: Input 1: Output	4
11	EXIO Signal Level	exio_data	ssssssssssssssss 15 . . . . . 0 n: 0 or 1 0: Input 1: Output	4
12	USB Function Status Flag	usb_function	on or off on: Enabled off: Disabled	15
13	Ether Status Flag	ether_enable	on or off on: Enabled off: Disabled	15
14	COM1 Status Flag	com1_enable	on or off on: Enabled off: Disabled	15
15	COM2 Status Flag	com2_enable	on or off on: Enabled off: Disabled	15
16	CF Status Flag	cf_enable	on or off on: Enabled off: Disabled	15

<Default Values>

No.	Item	Parameter Name	Initial value (Default)
1	I2C Status	i2c_enable	off
2	ADC Status	adc_enable	off
3	USB/IF Drive	usb_drv	C:¥
4	USB Host Status Flag	usb_host	on
5	Serial Host I/F Status	siohost_enable	off
6	Serial Host I/F COM Port	siohost_com	COM1
7	GPIO I/O Setting	gpio_select	0000000000000000
8	GPIO Output Drive Capability	gpio_drv	0000000000000000
9	GPIO Signal Level	gpio_data	0000000000000000
10	EXIO I/O Settings	exio_select	0000
11	EXIO Signal Level	exio_data	0000
12	USB Function Status Flag	usb_function	on
13	Ether Status Flag	ether_enable	on
14	COM1 Status Flag	com1_enable	on
15	COM2 Status Flag	com2_enable	on
16	CF Status Flag	cf_enable	on

14.7 Environment Variable Parameters

<Settings>

No.	Item	Parameter Name	Format and Setting Value	Area Length
1	Countdown Time in Seconds	cd_time	Numerical (in sec)	2
2	Time Difference•(UTC)	time_diff	+hhmm or -hhmm +hhmm: Positive time difference -hhmm: Negative time difference hh: 00 - 23 mm: 00 - 59	15
3	syslog Output Status	syslog	on or off on: Enabled off: Disabled	
4	User definition 1	user1	User definition character string	63
5	User definition 2	user2	User definition character string	63
6	User definition 3	user3	User definition character string	63
7	User definition 4	user4	User definition character string	63
8	User definition 5	user5	User definition character string	63
9	User definition 6	user6	User definition character string	63
10	User definition 7	user7	User definition character string	63
11	User definition 8	user8	User definition character string	63
12	User definition 9	user9	User definition character string	63
13	User definition 10	user10	User definition character string	63

<Default Values>

No.	Item	Parameter Name	Initial value (Default)
1	Countdown Time in Seconds	cd_time	3
2	Time Difference•(UTC)	time_diff	+0900
3	syslog Output Status	syslog	off
4	User definition 1	user1	(NULL)
5	User definition 2	user2	(NULL)
6	User definition 3	user3	(NULL)
7	User definition 4	user4	(NULL)
8	User definition 5	user5	(NULL)
9	User definition 6	user6	(NULL)
10	User definition 7	user7	(NULL)
11	User definition 8	user8	(NULL)
12	User definition 9	user9	(NULL)
13	User definition 10	user10	(NULL)

14.8 Dialing Control Parameters

<Settings>

No.	Item	Parameter Name	Format and Setting Value	Area Length
1	ISP Dial Number	isp_dial_no	Dial number numerical string (including special symbols)	31
2	ISP User ID	isp_user_id	ISP user ID string	47
3	ISP Password	isp_passwd	ISP password string	31
4	ISP PPP Communication Authentication Mode	isp_ppp_auth	chap or pap	
5	DNS Server Address for ISP	isp_dns_addr	xxx.xxx.xxx.xxx xxx: Numbers (000 - 255)	15
6	ISP Client IP Address	isp_self_addr		
7	SMTP Dial Number	smtp_dial_no	Dial number string (including special symbols)	31
8	SMTP User ID	smtp_user_id	SMTP user ID string	47
9	SMTP password	smtp_passwd	SMTP password string	31
10	SMTP PPP Communication Authentication Mode	amtp_ppp_auth	chap or pap	
11	SMTP DNS Server Address	isp_smtp_dns	xxx.xxx.xxx.xxx xxx: Numbers (000 - 255)	15
12	SMTP Client IP Address	smtp_self_addr		
13	PPP Local IP Address	ppp_local_addr	xxx.xxx.xxx.xxx xxx: Numbers (000 - 255)	15
14	PPP Remote IP Address	ppp_remote_addr	xxx.xxx.xxx.xxx xxx: Numbers (000 - 255)	15
15	PPP Subnet Mask	ppp_netmask	xxx.xxx.xxx.xxx xxx: Numbers (000 - 255)	15
16	Monitoring Status of PPP Communication	ppp_timeout	on or off on: Line blocked off: No line blocked	15

<Default Values>

No.	Item	Parameter Name	Initial value (Default)
1	ISP Dial Number	isp_dial_no	(NULL)
2	ISP User ID	isp_user_id	(NULL)
3	ISP Password	isp_passwd	(NULL)
4	ISP PPP Communication Authentication Mode	isp_ppp_auth	chap
5	DNS Server Address for ISP	isp_dns_addr	(NULL)
6	ISP Client IP Address	isp_self_addr	(NULL)
7	SMTP Dial Number	smtp_dial_no	(NULL)
8	SMTP User ID	smtp_user_id	(NULL)
9	SMTP password	smtp_passwd	(NULL)
10	SMTP PPP Communication Authentication Mode	smtp_ppp_auth	chap
11	SMTP DNS Server Address	isp_smtp_dns	(NULL)
12	SMTP Client IP Address	smtp_self_addr	(NULL)
13	PPP Local IP Address	ppp_local_addr	192.168.10.1
14	PPP Remote IP Address	ppp_remote_addr	192.168.10.2
15	PPP Subnet Mask	ppp_netmask	255.255.0.0
16	Monitoring Status of PPP Communication	ppp_timeout	on

14.9 Available Regular Expressions

The following table shows a list of regular expressions that can be used with regular expression functions in WSC script:

No.	Regular Expression	Description
(1)	.	matches any one character. [Sample Code] “a.c”: “abc”, “a5c”, etc.
(2)	*	matches zero or more occurrences of one character. [Sample Code] “a*c”: “ac”, “aaac”, “aaaaaaac”, etc.
(3)	[ ]	matches any one of the characters within the brackets ([]). [Sample Code] “[]”: “ac”, “aaac”, “aaaaaaac”, etc.
(4)	-	specifies the range of continuous characters used in conjunction with the bracket of (3). [Sample Code] [0-9]: numbers 0 - 9 are the matching target. [A-Z a-z]: A - Z and a - z are the matching target.
(5)	^	(a) negates the subsequent character or set of characters when used in a [ ] or at any position other than the beginning of a line. [Sample Code] [^0-9]: Numbers other than 0-9 are the matching target. (b) anchors the match to the beginning of a line when used at the beginning of a line.
(6)	{m,n}	indicates m to n times occurrence of preceding character. [Sample Code] 5{3}: “5” occurs three times in a row. G{4,}: “G” occurs more than three times in a row. M{2,5}: “M” occurs more than two times but up to 5 times in a row.
(7)	$	anchors the match to the end of a line. [Sample Code] a$: the line ends with ‘a.’
(8)	+	matches one or more occurrences of a character that precedes the expression (pattern.) [Sample Code] [A-Z]+: a string made up of letters A-Z, such as “HELLO.”
(9)	?	matches zero or more occurrence of a character that precedes the expression (pattern.) [Sample Code] “WSC ?module”: 0 or one space exists, such as “WSCmodule” and “WSC module.”
(10)	|	indicates a choice between multiple items. [Sample Code] “ABC|DEFG|HI X”: one of “ABC X”, “DEFG X”, or “HI X” is matching target.
(11)	()	marks the beginning and the end of subexpression. [Sample Code] “(ABC|XY) (10|20|30)”: a string made up of three parts; the first part is “ABC” or “XY”, second part is a space, the last part is “10”, “20”, or “30”.

14.10 Detailed Information about Error Codes

The following table provides detailed information on WSC script function error codes:

Category/Error Code	Description
(1) Time Functions	
- SNTP	
0xFF090001	Invalid parameter
0xFF090002	Failed to connect to the server
0xFF090003	Invalid SNTP flag
0xFF090004	Failed to generate a task
- RTC	
0xFF220001	Invalid parameter
0xFF220002	Driver uninstallation
0xFF220003	Driver has been registered
(2) SIO I/O Functions	
- UART(COM2)	
0xFF190065	Not initialized
0xFF190066	Receive buffer overflow
0xFF190067	Parity error
0xFF190068	Overrun error
0xFF190069	Framing error
0xFF19006A	Break detected
0xFF19006B	Receive FIFO error
0xFF19006C	Transmit buffer overflow
- Serial (COM0 or COM1)	
0xFF210001	Invalid parameter
0xFF210002	Timed out
0xFF210003	Failed to sleep the task
0xFF210004	Not initialized
0xFF210005	Receive buffer overflow
0xFF210006	Parity error
0xFF210007	Overrun error
0xFF210008	Framing error
0xFF210009	Break detected
(3) PIO I/O Functions	
0xFF230001	Invalid parameter
0xFF230002	Failed to generate a task
0xFF230003	Failed to generate a semaphore
0xFF230004	Failed to generate a queue
0xFF230005	Not initialized
0xFF230006	No queues available
(4) AD Functions	
0xFF250001	Invalid parameter
0xFF250002	Failed to generate a semaphore
0xFF250003	Timed out while retrieving data
0xFF250004	Not initialized
(5) IIC I/O Functions	
0xFF240001	Parameter Abnormality
0xFF240002	Driver not yet installed
0xFF240003	Failed to generate a semaphore
0xFF240004	Not initialized
0xFF240005	Busy
0xFF240006	Non-acknowledged response detected
0xFF240007	Lost arbitration
(6) FTP I/O Functions	
0xFF030001	Failed to generate a task
0xFF030002	Failed to open a new session
0xFF030003	Failed to connect
0xFF030004	LOGIN command failed
0xFF030005	Failed to set a PORT
0xFF030006	TYPE command failed
0xFF030007	STORE command failed
0xFF030008	Local file already exist
0xFF030009	Failed to open a file
0xFF03000A	Failed to read a file
0xFF03000B	Failed to delete a file
0xFF03000C	Invalid parameter
0xFF30000D	STOR command failed
0xFF30000E	Failed to generate a semaphore
0xFF30000F	Not initialized
0xFF300010	Failed to generate a task
0xFF300011	Busy

Category/Error Code	Description
(7) Send E-Mail Functions	
0xFF060001	Failed to initialize
0xFF060002	Invalid parameter
0xFF060003	Busy
(8) Event Handling Functions	
- TIMER Events	
0xFF0E0001	Invalid parameter
0xFF0E0002	Failed to generate a semaphore
0xFF0E0003	Not initialized
0xFF0E0004	Failed to generate a task
0xFF0E0005	Failed to generate a queue
0xFF0E0006	No queues available
- RTC Events	
0xFF0C0001	Invalid parameter
0xFF0C0002	Failed to generate a task
0xFF0C0003	Failed to generate a semaphore
0xFF0C0004	Failed to generate a watchdog
0xFF0C0005	Failed to access RTC
0xFF0C0006	Failed to generate a queue
0xFF0C0007	Not initialized
0xFF0C0008	No queues available
- SIO Events	Refer to error codes for SIO I/O functions.
- PIO Events	Refer to error codes for PIO I/O functions.
- PIO PULSE Events	Refer to error codes for PIO I/O functions.
(9) System Control Functions	
- Parameter Control	
0xFF290001	Invalid parameter
0xFF290002	Writing to Flash ROM failed
0xFF290003	Reading FlashROM failed
0xFF290004	Failed to generate a semaphore
0xFF290005	Not initialized
0xFF290006	Invalid parameter length
0xFF290007	Invalid parameter value
- Event Log Output	
0xFF140001	Invalid parameter
0xFF140003	Not initialized
(10) Dialing Control Functions	
0xFF110001	Invalid parameter
0xFF110002	Connection failed
0xFF110003	AT command failed
0xFF110004	Receive time out
0xFF110005	Already running
0xFF110006	Failed to generate a task
0xFF110007	Failed to generate a semaphore
0xFF110008	COM driver Not Ready
(11) Relation of Network	
- DNS	
0xFF040001	Parameter Abnormality
0xFF040002	Response Error
0xFF040003	Response Type Error
0xFF040004	Response Character String Error
0xFF040005	Initialization Failure

14.11 Terminal Software

This section guides you through a simple way of using terminal software to establish a serial connection between ProDigio and a personal computer to use the debug monitoring function.

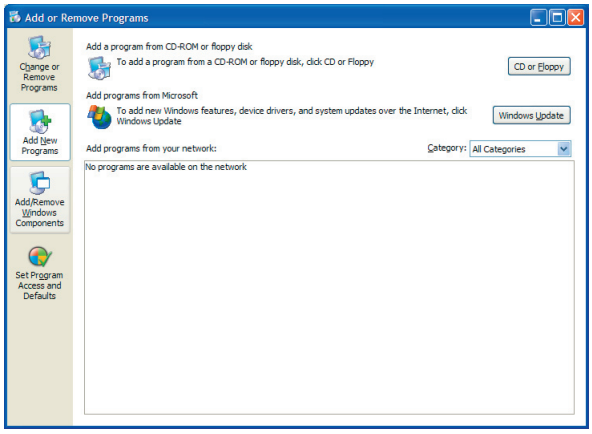
Note: This section uses Hyper Terminal as an example because it is a standard Windows bundled software.
For more information on how to use terminal software of your choice, see the manual provided with your terminal software because usage of terminal software can differ depending on OS or the terminal software being used.

14.11.1 Connecting to Hyper Terminal

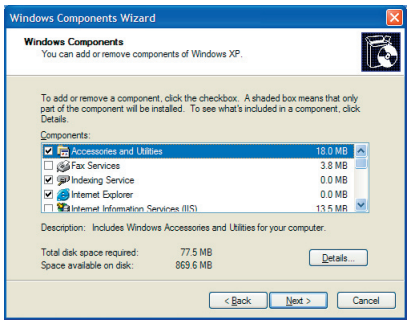
The Hyper Terminal program icon can be found in **Start** --> **All programs** --> **Accessories** --> **Communications** --> **Hyper Terminal**.
If Hyper Terminal is not installed on your computer, perform the following steps to install it:

1. Open **Add/Remove Programs** in **Control Panel**.

2. Click **Add/Remove Windows Components**.


Add or Remove Programs
Change or Remove Programs
Add New Programs
Add/Remove Windows Components
Set Program Access and Defaults
Add a program from CD-ROM or floppy disk
To add a program from a CD-ROM or floppy disk, click CD or Floppy
CD or Floppy
Add programs from Microsoft
To add new Windows features, device drivers, and system updates over the Internet, click Windows Update
Windows Update
Add programs from your network:
Category:
All Categories
No programs are available on the network

3. Click **Accessories and Utilities**, and then click **Details**.


Windows Components Wizard
Windows Components
You can add or remove components of Windows XP.
To add or remove a component, click the checkbox. A shaded box means that only part of the component will be installed. To see what's included in a component, click Details.
Components:
Accessories and Utilities 18.0 MB
Fax Services 3.8 MB
Indexing Service 0.0 MB
Internet Explorer 0.0 MB
Description: Includes Windows Accessories and Utilities for your computer.
Total disk space required: 77.5 MB
Space available on disk: 869.6 MB
Details...
< Back
Next >
Cancel

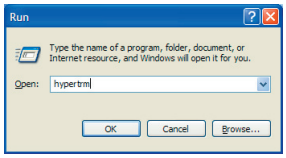
4. Click **Communications**, and then click **Details**.
5. Select **Hyper Terminal** check box, and click **OK**.

Note:
To perform these steps on Windows NT, 2000, or XP, you need to login using an account with administrative permission for the local computer.

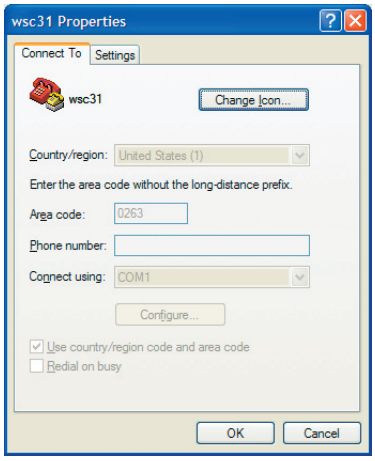
Please note that actual steps and images differ depending on the OS running.

14.11.2 Creating a New Connection

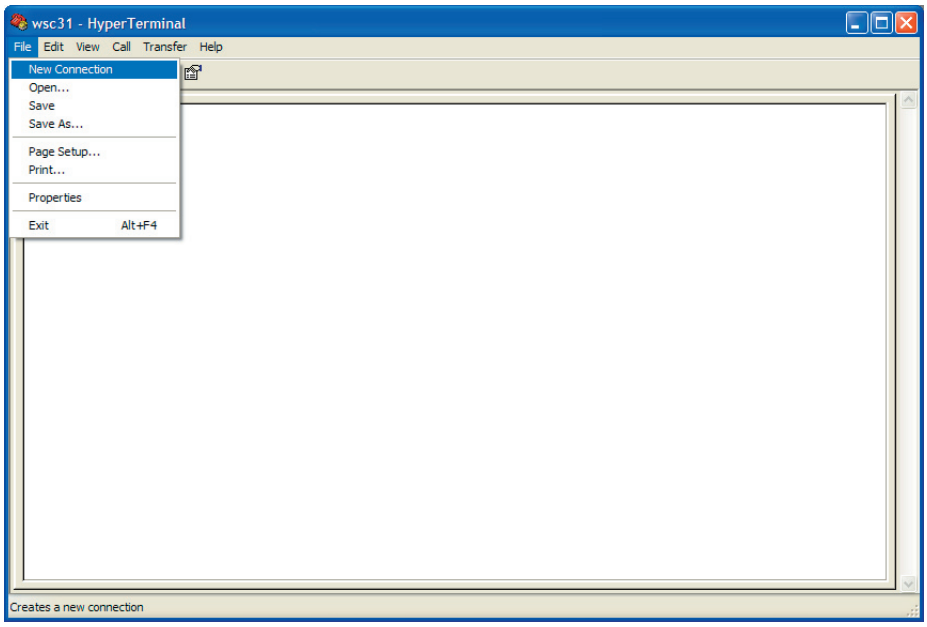
1. Open Hyper Terminal from **Start** menu or by entering **hypertrm.exe** in **Run** window



2. In the **Location Information** dialog box, enter your local information and then click **OK**.

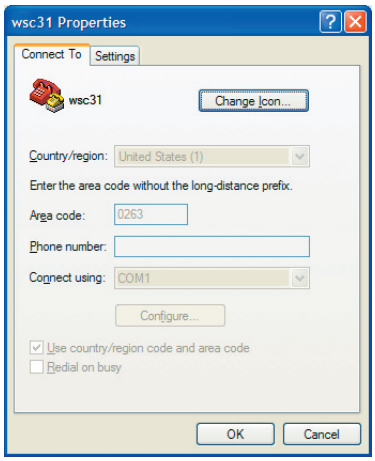


3. On **File** menu, click **New Connection**.

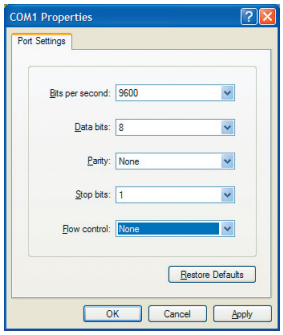


4. Enter a **name** for the connection in Name box. (For example, enter **WSC Connection.**) Select an appropriate **icon** from Icon box, and then click **OK**.

5. From the list of **Connect using** drop-down box in the **Connect to** dialog box, select the port to use. (For example, select COM1 serial port.)



6. In the properties window for the COM you selected in step 5, change the settings as follows:



Baudrate (bits per second): 9600

Data bits: 8

Parity: None

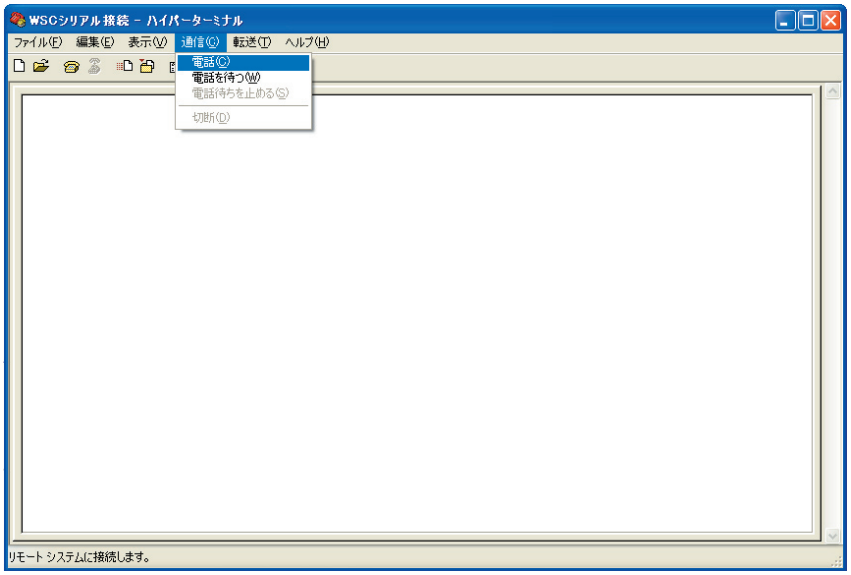
Stop bits: 1

Flow control: None

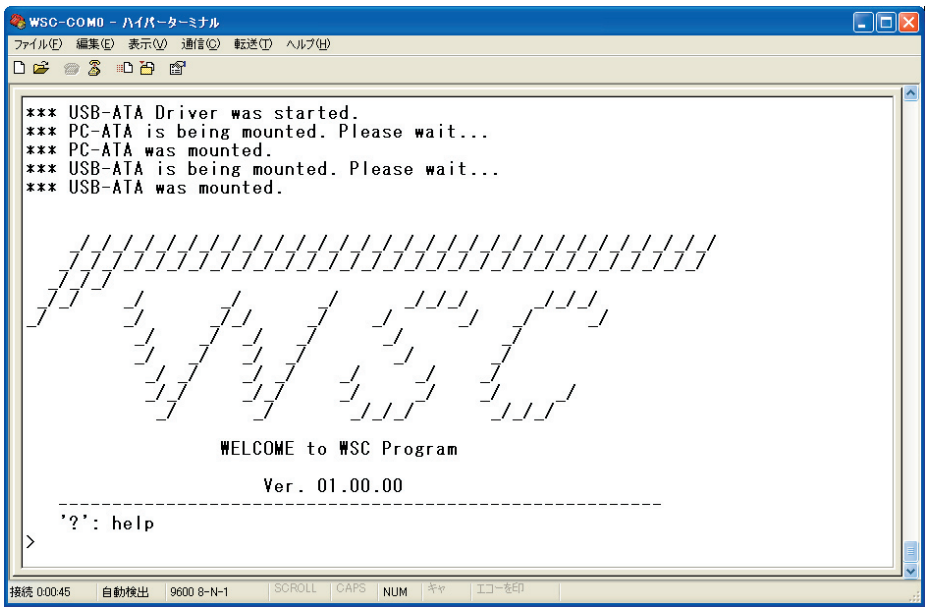
7. The following console window is displayed. In this window, you can type ProDigio AT commands to communicate with ProDigio.

   To open communication with ProDigio, click **Call** command from **Call** menu.

   For ProDigio terminal commands, see “14.12 List of Debug Monitoring Commands.”

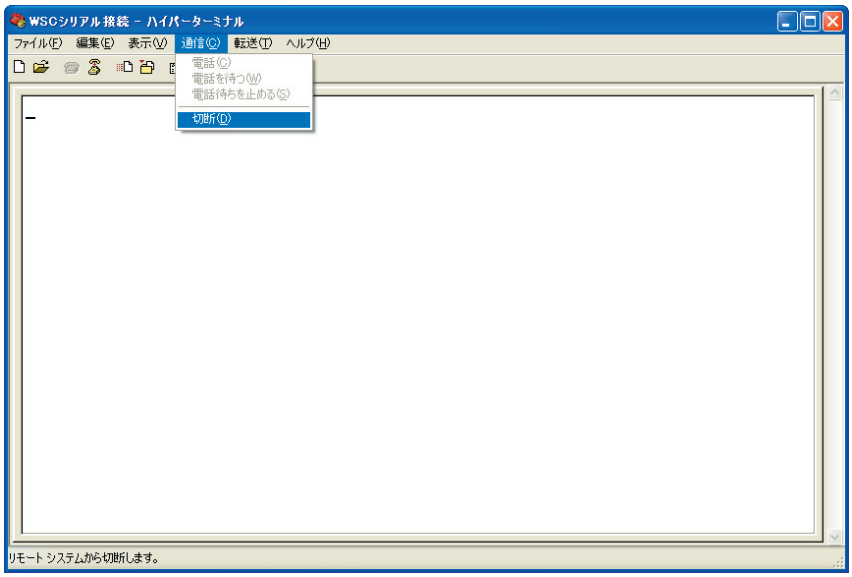


8. If you are using the ProDigio Training Kit, click the **Reset** button and restart ProDigio. When the connection between ProDigio and your PC is established, the following screen is displayed:



14.11.3 Closing Hyper Terminal

1. Click the **Disconnect** button or **Disconnect** command from **Call** menu to end the connection.



2. Exit Hyper Terminal.

   If you try to exit Hyper Terminal while the connection is still active, you will receive the message “Are you sure you want to disconnect now?” Click **Yes** to exit Hyper Terminal.

14.12 List of Debug Monitoring Commands

No.	Command	Command Syntax	Description
1	?	?	Display help screen
2	help	help	Display help screen
3	reboot	reboot	Restart
4	evtdump	evtdump	View event log
5	evtshow	evtshow	View event log
6	evtclr	evtclr	Clear event log
7	date	date	Show date and time
8	bye	bye	Close DiagPort line
9	sntp	sntp	Start SNTP
10	sysup	sysup	Update firmware
11	exec	exec "script_path"	Execute the script
12	prmhelp	prmhelp	Display parameters help screen
13	prmshow	prmshow [type]	Show parameters
14	prmboot	prmboot [type]	Set boot parameters
15	prmnet	prmnet [type]	Set network parameters
16	prmserial	prmserial [type]	Set serial parameters
17	prmdial	prmdial [type]	Set dial parameters
18	prmenv	prmenv [type]	Set environment parameters
19	prmio	prmio [type]	Set I/O parameters
20	prmfw	prmfw [type]	Create a parameter file
21	prmini	prmini	Initialize parameters
22	netshow	netshow ifno	Show network information
23	pppshow	pppshow	Show PPP information
24	ls	ls ["path"]	Show the list of directories
25	cd	cd "path"	Change the current directory to path
26	pwd	pwd	Show the current directory

14.13. Debug Monitoring Functions

Your WSC script may not work properly if the script contains any inappropriate statements.
In such cases, you will be able to identify the cause and the point of failure by debugging your script. With ProDigio, you can debug the script from the terminal software. This section provides some tips for debugging.

14.13.1 Standard Output

Generally, ProDigio uses COM0 (terminal software console) as its standard output.
If you are connecting to the debug port through Telnet (explained later in this section), Telnet' s port is used as the standard output.

14.13.2 Error Messages

If a script contains erroneous statements, you will see error message(s) on the terminal software console. The error message tells you which part of the script is causing the error.
For more information about error messages, see “14.13.6 List of Debug Error Messages.”

14.13.3 Event Log

ProDigio writes the progress of operations in an event log. This log file can hold up to 1364 entries. When the file reaches the limit, older entries are overwritten by new entries.
To view the log, run the **evtdump** command from terminal software console. You can also view the log file by running this command via Telnet.
The syntax is “evtdump number_of_entries.”

```
>evtdump          --> display all logs
>evtdump 100      --> display the latest 100 logs
```

The **error** function and the **outevtlogmsg** function can be used to output new entries to the event log.
These functions are not only capable of recording error handling events but also managing the operation from remote by outputting the operation status to the log file.

14.13.4 Referring to WSC Configuration Parameters

There are three types of WSC configuration parameters:

1. Boot parameter value (0)
2. Current value (1)
3. Factory default value (2)

To view these parameters, use the **prmshow** command in one of the following ways that suits your needs:

>prmshow 0 --> display boot parameter value
>prmshow 1 --> display the current value
>prmshow 2 --> display the factory default value

You can specify 0, 1, or 2 with the following commands to set the boot parameter value, the current value, or the factory default value, if necessary:

prmboot, prmnet, prmserial, prmdial, prmenv, prmio, prmfw

Note: if you do not specify a configuration parameter, it defaults to 0 (boot parameter.)

14.13.5 Connecting to Debug Port via Telnet

If you connect to port 8002 through Telnet, the standard output is switched to the Telnet port. That is, you are going to use the Telnet console as the terminal software.

When connecting to ProDigio through Telnet, you need to provide the login user ID and password.

14.13.6 List of Debug Error Messages

No.	Error Message	Description
1	Script not Found	Could not find the script.
2	Function not Found	Could not find the function.
3	Function error	Function error.
4	Argument error	An argument with no value was found.
5	Too few Argument	There are not enough arguments.
6	Argument format error	The first argument is not specified in regular expression.
7	Sentence error	Syntax error
8	Variable is not defined	The variable is not defined.
9	Declaration overlaps	Duplicate declaration exists.
10	Index error	The index is invalid.
11	Parenthes error	An opening or closing parenthesis is missing.
12	Quart error	An opening or closing quotation mark (“) is missing.
13	‘if’ and ’else’ do not correspond	if-else statement correspondence error.
14	Divide by ZERO	Dividing by zero is an invalid operation.
15	Illegal operator	The operator is invalid.
16	Illegal operator to string	An illegal operator is used in the string.
17	Illegal operator to hash	An illegal operator is used in the associative array.
18	File open error	Could not open the file.
19	Alloc failed	Failed to allocate memory.
20	Invalid discriptor	The descriptor is invalid.

<Message Format>

<<WCR>:Function error:[*Script File Name*].[*Function Name*]:[*Function Name*]:Argument error at [*Script File Name*]. [*Function Name*]:(*line number in which an error occurred*) : [*Statement*]

Index

A

A/D Conversion ---------- 186
Access ---------- 133
access ---------- 2,24,47,57,60,76
Access Position in File ---------- 143,144
Algebra ---------- 25
algebraic operation ---------- 25
AND ---------- 26
application ---------- 7
arg[ ] Array ---------- 46
argument ---------- 42
arithmetic shift ---------- 27
array ---------- 18
array number ---------- 61
Array Operation ---------- 19
ASCII code ---------- 27,41,97
associative array 18,19,24,46,76,85,86,87,88,89,108,109,110,111,112,113,114,115,116,117,118,119,120,137,138,160,190,191,207,222,244
AT command ---------- 77,216,217

B

beep ---------- 152,153
binary ---------- 56
binary file ---------- 56,60
bit ---------- 16,25
Bit Operators ---------- 26
bit OR operation ---------- 26
bit string ---------- 27,170,171,173,177,180,185
boot parameter ---------- 243
branch process ---------- 28
Branch Structure ---------- 31
break ---------- 37
break statement ---------- 37,39
browser ---------- 50
buffer ---------- 60,61,159,162

C

camera ---------- 7
CGI ---------- 2,45,46,47,52,68,71,73,221
Change File Attributes ---------- 145
character ---------- 65
character code ---------- 20
client ---------- 47,48
Closes a directory ---------- 130
Closes a file ---------- 134
code ---------- 20
Code Block ---------- 13
command ---------- 2,11,31
comment ---------- 13,39
communication speed ---------- 158
comparison operator ---------- 27
computer ---------- 7,8,9
condition ---------- 27
conditional expressions ---------- 27
continue statement ---------- 37
control ---------- 2,31,37,47,167
Copies a file ---------- 142
Copy File ---------- 142
create a file ---------- 132
curly bracket ---------- 12,14
current directory ---------- 57,125,126,127,128,129,132,133,140,141,142,145,241
current path ---------- 125

D

data format ---------- 18,22
date/time ---------- 54
decimal number ---------- 15,16,26
declaration ---------- 19,21,22,222,244
definition ---------- 231
delete a file ---------- 234
development module ---------- 10
device ---- 2,10,47,71,72,77,187,188,189,190,191,208,220
Dial-up ---------- 217
dial-up ---------- 212,213,216,219
directory ---------- 9,10,125,126,241
disconnection ---------- 213
do..while statement ---------- 35
double quotation ---------- 17,19
Drive ---------- 53
driver ---------- 2,7

E

E-Mail ---- 77,195,221,235
E-mail ---- 67,72,73,196
e-mail ---- 39
element ---- 18,19,22,23,24,61,78,109
else statement ---- 31,33,72,222,244
enlarge ---- 18
Environment Variable Parameter ---- 231
Error ---- 104,131,234,235,242,244
error ---- 30,57,60,75,79,90,234
Error Message ---- 79,222,242,244
error message ---- 75
Escape Character ---- 17,40
Event Log ---- 210,235,242
event log ---- 77,79
Exclusive OR Operation ---- 26
Expansion ---- 17
Expression ---- 30
expression ---- 27,29,32,33,35

F

File ---- 55,56,57,58,60
File Size ---- 139
file size ---- 76
file system ---- 57
Flash ROM ---- 235
flash ROM ---- 77,204
Floating Point ---- 80,81
floating point ---- 75,82,84
FLOAT type ---- 15,16,80
Flowchart ---- 31
for statement ---- 31,36
FTP ---- 10,192,193,194,221,234
function ---- 12,13,20

G

gethashcount Function ---- 46
gethashcount function ---- 45

H

hardware ---- 7,10,208
hash ---- 18,24,61,62
hash buffer ---- 61
hexadecimal notation ---- 16,17
home security ---- 7
HTML ---- 53
HTML language ---- 47
HTTP server ---- 2,53
HyperTerminal ---- 10

I

I/O ---- 76,149,154,168,172,173,176,177,182,183,184,185,187,221,229
I/O connector ---- 2
if statement ---- 27,32,38
IIC ---- 77,187,188,189,190,191
index ---- 18,22
initialization ---- 77,215,216
INTEGER type ---- 22,25,26,27,30,80
interface ---- 2,47
Internet ---- 2

J

join ---- 17

K

key ---- 18,24,61,110,111,112,113,114
keyword ---- 28

L

LAN ---- 2
line break ---- 12,40
link ---- 53
logical operation ---- 28
logical operator ---- 28
logical shift ---- 27
loop ---- 22,36
lower case ---- 75,106,107

M

MAC address ---- 77,207
matching ---- 38,39,122,123
memory ---- 18,20,21,22,55,222,244
meta character ---- 65
mobile phone ---- 2
multidimensional array ---- 18,22,23

N

nest ---- 221
network ---- 2,241
network parameter ---- 241
number ---- 9,15,16,20

O

octal number ---- 15,16
Opens a directory ---- 129
Opens a file ---- 133
operating system ---- 7
operator ---- 25,27
OR ---- 26,39
order ---- 9,12,22
OR operation ---- 26
OS ---- 7
overwrite ---- 60

P

pattern ---- 65,78,122
PIO ---- 168
priority ---- 29,70,71
ProDigio ---- 2,7,8,9,10,11,12,14,22,27,28
ProDigio web site ---- 1
programming ---- 2,7,8,18,22,31,42,47
program source ---- 47
provider ---- 212,213
pseudo language ---- 31
pulse ---- 70
punctuation ---- 9

R

reboot ---- 201,241
Recursive Structure ---- 31
reference ---- 24,48
regular expression ---- 65,122,123,124,222,233
remote ---- 2,7,47,192,193,232,242
request ---- 53,161,192,193
result code ---- 77,219,220
return value ---- 42

S

sample ---- 14,44,46,60
sample script ---- 44,46,50,58,61,72,73,78
search ---- 28,57
semicolon ---- 12,14
sendmail function ---- 72,73
send an E-mail ---- 72,73
sequential structure ---- 31
serial ---- 10,52,69
serial communication ---- 52,53
serial connection ---- 10,236
serial parameter ---- 227,241
series of statement ---- 11,42
shift ---- 27
shift operation ---- 27
SIO ---- 52,69,76,154,198,199,200,234,235
size ---- 9,12,18,27,60,136
SMTP ---- 72
smtp ---- 77,196,212
software ---- 7,8,10
source ---- 30,47
space ---- 12,13
SSI ---- 2,47,48,51,52,68,71,73
statement ---- 32,33,34,35,36,37,38
store ---- 18,19,22
string ---- 17,25,30,32,39,42
STRING type ---- 30
structure ---- 31
substitution ---- 25,30
suffix ---- 18,22,24
switch statement ---- 38,39
symbol ---- 9,25

T

T&D ---------- 1,249
tab ---------- 12
Telnet ---------- 242,243
terminal ---------- 10,11,40,69
terminal command ---------- 239
terminal software ---------- 10,11,236,242,243
text file ---------- 9,56,58
time ---------- 51,54,68
timeout ---------- 52
trigger ---------- 52
type ---------- 15,16,20
type conversion ---------- 30

U

unary ---------- 25,29
unary operators ---------- 25,29
underscore ---------- 21,24
upper case ---------- 1,75,106,107
USB ---------- 10,208,229

W

wcr ---------- 10,11,45,50,51,68
Web ---------- 2
Web browser ---------- 2,77
while statement ---------- 34
WSC script ----------
2,8,10,12,14,15,17,18,22,25,26,27,28,42,46,47,50,73

Support and Customer Service

T&D Corporation

Sasaga 5652-169, Matsumoto, NAGANO 399-0033 Japan
Facsimile: (+81) 263-26-4281

[ProDigio Website]
Please visit the ProDigio web site for what's new with our product, event information, obtaining software and support service:

http://english.prodigio.jp/

overseas@tandd.co.jp

WSC Script User's Manual

February 2006, First Edition

Published by T &D Corporation

© Copyright 2006 T&D Corporation.Allright reserved.